DETImótica

Universidade de Aveiro
Departamento de Eletrónica, Telecomunicações e Informática

Projeto em Engenharia Informática
Relatório Técnico

Eurico Dias, Gonçalo Perna, João Trindade, Pedro Valério, Ricardo Carvalho, Rodrigo Rosmaninho

Professor Diogo Gomes, Professor José Vieira

09 de junho de 2020

DETImótica

Relatório Técnico de Projecto em Engenharia Informática do Mestrado Integrado em Engenharia de Computadores e Telemática da Universidade de Aveiro

Autoria de Eurico Dias (72783) dias.eurico@ua.pt, Gonçalo Perna (88823) goncaloperna@ua.pt, João Trindade (89140) jatt@ua.pt, Pedro Valério (88734) pedrovalerio@ua.pt, Ricardo Carvalho (89147) rapc99@ua.pt, Rodrigo Rosmaninho (88802) r.rosmaninho@ua.pt

Orientado pelo Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro, Diogo Gomes, e pelo Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro, José Vieira.

Resumo

O projeto DETImótica visa a transformação do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro num espaço mais inteligente e moderno, através da instalação de vários sensores ambientais nas suas divisões. Com estes, a comunidade pode interagir abertamente de uma forma que garante expansibilidade futura por parte do corpo discente e docente, cuja diversidade de conhecimentos, habilidades, e interesses é especialmente apropriada a este domínio.

O objetivo principal foi a implementação de um backend agregador robusto e de interfaces programáticas abrangentes e de utilização acessível que permitem, entre outros, a gestão e obtenção de informação sobre os recursos instalados, a consulta de dados de telemetria sob variadas formas, e o controlo granular de permissões dos utilizadores.

A equipa procurou igualmente estabelecer e implementar a estratégia de sensorização, comunicação, gestão e segurança. Assim como a arquitetura de dados, o sistema de notificações e alertas, e a documentação dos diversos módulos.

Foi também desenvolvida uma plataforma de gestão dos recursos instalados e dos acessos aos mesmos destinada a administradores e alguns exemplos de aplicações interativas que podem ser construídas utilizando o sistema. São exemplos as dashboards para visualização dos dados de telemetria e uma aplicação móvel de acesso fácil que inclui notificações push.

Com todas estas bases concluídas e validadas, e já que o sistema foi construído de forma a maximizar a liberdade de desenvolvimento futuro, qualquer membro da comunidade pode agora instalar um novo conjunto de sensores ou integrar a plataforma numa aplicação da sua autoria.

Agradecimentos

Em virtude de todo o apoio prestado durante a conceptualização e desenvolvimento do projeto, o grupo gostaria de estender agradecimentos aos professores orientadores, aos professores da Unidade Curricular de Projeto em Engenharia Informática, ao Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro e ao Instituto de Telecomunicações.

Índice

Índice de Tabelas

Índice de Figuras

Glossário

ABAC	Attribute-based access control
ADC	Analog-to-digital Converter
AES	Advanced Encryption Standard
API	Application Programming Interface
CSS	Cascading Style Sheets
CPU	Central Processing Unit
DETI	Departamento de Electrónica, Telecomunicações e Informática
FirebaseCM	Firebase Cloud Messaging
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
I2C	Inter-Integrated Circuit
IdP	Identity Provider
ISU	Integrated Sensor Unit
IoT	Internet of Things
InfluxQL	Influx Query Language
IT	Instituto de Telecomunicações
JSON	JavaScript Object Notation
MQTT	MQ Telemetry Transport
OAuth	Open Authorization
PDP	Policy Decision Point
REST	Representational State Transfer
SQL	Structured Query Language
sTIC	Serviços de Tecnologias de Informação e Comunicação
UA	Universidade de Aveiro
UC	Unidade Curricular
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
UUID	Universally Unique Identifier
VOC	Volatile Organic Compound
VPN	Virtual Private Network
WSGI	Web Server Gateway Interface
WSO2	Web Services oxygenated

1 Introdução

O presente relatório técnico visa descrever e analisar os detalhes e processo de implementação do projecto DETIMótica. A grande motivação por trás deste projecto foi a possibilidade de conseguir monitorizar as condições presentes e passadas do espaço do departamento e, no futuro, da universidade, de modo a poderem ser utilizadas para diversos fins. Para tal, foi adoptada a instalação de módulos de sensorização em várias salas.

Este projecto é uma reimplementação de um projecto passado, cujo tema tem alguns precedentes que serão apresentados numa das secções seguintes.
Uma característica importante e grande preocupação ao longo do desenvolvimento de todo o projecto foi a disponibilização de um meio de interação com os dados sensoriais aberto a toda a comunidade. Com a criação de uma API (*Application Programming Interface*), visou-se, não só estender a usabilidade do projecto, permitindo que desenvolvedores externos possam complementar e estender o trabalho existente, como abstrair o backend de implementação dos módulos de sensorização e base de dados, garantindo uma facilidade de uso.

O resultado final foi uma API que permita o acesso às várias funcionalidades do sistema, módulos de instalação de sensores, uma gateway para receber e encaminhar a informação dos módulos de sensorização, uma plataforma de gestão de acessos, dashboards para visualização da informação e uma aplicação móvel, disponível a todos os utilizadores.

O resto deste relatório irá começar por contextualizar o projecto, dar conta de outros trabalhos e progressos semelhantes ou relacionados, apresentar uma conclusão do nosso projecto e seguindo-se uma análise dos requisitos do sistema. Seguidamente, serão explicados os detalhes e processo de implementação de cada uma das componentes fundamentais do projecto, bem como outros aspectos relacionados aos mesmos. Finalmente, serão discutidos, em mais detalhe, os resultados obtidos neste projecto, problemas, limitações e funcionalidades implementadas, inicialmente não previstas nos planos.

1.1 Contexto

Devido aos dispositivos eletrónicos terem vindo a ficar cada vez mais acessíveis, as soluções com base nos mesmo são cada vez mais viáveis.

Neste caso pretendemos modernizar o DETI, especialmente a nível da monitorização das salas, através da instalação de sensores de várias categorias como por exemplo, sensores de humidade.

2 State of the Art

Sempre foi uma das grandes imagens de marca da Universidade de Aveiro a vertente humanitária bem como inovação e aposta tecnológica. No seguimento desses dois conceitos, o DETI encontra-se no início da sua modernização enquanto conjuga ambos.

Com isto em mente, a facilidade de instalar dispositivos que permitam coletar, processar e comunicar informações é algo imprescindível. No ano transato, o MakerLab foi escolhido como prova de conceito para a instalação de uma câmara, pretendendo-se agora equipar mais divisões com aparelhos relevantes para o ecossistema. Alguns desses aparelhos já estão presentes na universidade, não estando no entanto instalados e/ou configurados corretamente.

A página api.web.ua.pt permite visualizar algumas das APIs disponibilizadas pela instituição, sendo esperada a adição de novas APIs, criadas pelo corpo docente e discente, almejando a utilização dos sensores em futuras iniciativas.

2.1 Projetos Relacionados

2.1.1 Smart Sensors

A Universidade de Alicante, fundada em 1979 em Espanha, tem efetuado uma forte aposta no conceito de *Smart City*, resultando na evolução do mesmo para *Smart University*. Com o intuito de fornecer novas soluções para o resto do mundo ao fortalecer a componente da inovação tecnológica, a instituição espera melhorar a qualidade de vida da sua população, ao mesmo tempo que melhor a sua própria sustentabilidade.

Os SmartLabs são os laboratórios visados por esta geração de serviços inteligentes, entre os quais estão os Smart Sensors. Um Smart Sensor é caracterizado por obter informações sobre o ambiente em que se encontra, conseguindo armazenar, processar e enviar as mesmas. É por isso um elemento crucial para tornar um espaço inteligente.

Dentro deste projeto, os dispositivos foram divididos em duas categorias: sensores gerais e sensores específicos. Na primeira, estão incluídos aparelhos que consigam calcular a temperatura, luminosidade, humidade, etc. Aparelhos

que realizem medições e enviem as mesmas. Já na segunda, uma conexão à internet é necessária e a massa é o alvo da análise. Desde giroscópios, passando por sensores de posição como o GPS e incluindo ainda câmaras que reconheçam imagens.

2.1.2 IOTwente

No início do ano de 2016, a Universidade de Twente na Holanda aprovou o projeto "Living Smart Campus". Com o campus a ser considerado um autêntico laboratório vivo, foi considerado a escolha ideal para seguir em frente com um projeto que pretende tornar a instituição cada vez mais sustentável e progressista.

Uma das áreas do projeto, é a IOTwente, cuja finalidade passa por interligar diversos dispositivos, ao mesmo tempo que estes se tornam *wireless* e autónomos. A ambição do projeto passa por iniciar esta modernização num simples termóstato, podendo alcançar algo como a comunicação entre veículos.

O Kennispark Twente, ligado à investigação da Universidade de Twente, é uma das grandes inspirações do IOTwente, sendo possível encontrar neste informações provenientes de sensores relativas ao consumo de eletricidade, qualidade do ar e vibrações em edifícios.

2.2 Conclusão

A importância do meio ambiental é algo que está a crescer de mãos dadas com as universidades pioneiras, suportada por uma forte aposta na inovação tecnológica. Tanto a universidade de Aveiro, como a de Alicante e Twente são universidades jovens, fundadas após 1960, sendo que todas preenchem os requisitos referidos.

A tenra idade destas instituições, faz com que estes projetos estejam todos na sua fase inicial, sendo no entanto um marco importante e de destaque. Um dos grandes objetivos desta aposta, é fornecer às massas presentes nestes ambientes uma qualidade de vida superior e mais detalhada, ao mesmo tempo que a sustentabilidade incrementa.

Com o prestígio que estas inovações trazem, advém um forte senso de responsabilidade. É, por isso, necessário garantir a segurança dos dados e das comunicações orientes destes meios inteligentes, não podendo ser descuidado o facto de que é uma área em constante evolução, sendo necessária atenção à mesma regularmente.

3 Análise Conceptual

3.1 Requisitos do Sistema

Esta secção apresenta a especificação dos requisitos do sistema, sendo apresentado nas próximas sub-secções a descrição dos processos realizados para o levantamento de requisitos, seguida da lista de atores e diagramas de casos de utilização, assim como requisitos funcionais e não funcionais.

3.1.1 Levantamento de Requisitos

No âmbito da realização do projeto, a pesquisa da equipa focou-se em perceber quais seriam os dados mais relevantes à equipa de administração do DETI e da Universidade de Aveiro, abordando para isso, maioritariamente, o orientador do projeto: professor Diogo Gomes.

No âmbito do desenvolvimento da API e da sua integração no WSO2, assim como da utilização do IdP-UA, a equipa recolheu requisitos com os sTIC e com o orientador, o professor Diogo Gomes.

A consulta da wiki relativa ao inventário disponível no MakerLab serviu para fazer um levantamento do material já disponível mas não utilizado no departamento, percebendo melhor quais as métricas que irão ser monitorizadas. A consulta da documentação dos sensores e dos protocolos que irão ser utilizados permitiu obter um conjunto de requisitos não-funcionais relativos aos intervalos de medição e segurança nas comunicações.

3.1.2 Atores

Na tabela 3.1 são apresentados os atores do sistema juntamente com o seu papel. Podemos considerar que o Utilizador apresenta um papel de destaque, podendo este utilizar o projeto para obter informações ou desenvolver novos projetos, uma vez que o projeto DETImótica tem por um lado o objetivo de disponibilizar informações para utilizadores do DETI, e por outro lado tem o objetivo de simplificar novos projetos que pretendam utilizar a API quer a nível

do desenvolvimento de módulos de sensorização, quer a nível de aplicações que consumam os dados produzidos por estes módulos.

Ator	**Papel no sistema**
Identity@UA	Tecnologia que permite a autenticação dos utilizadores pertencentes à Universidade de Aveiro ao nível central.
Administrador	Os administrador pode consultar os dados dos sensores na dashboard e gerir os acessos na plataforma de gestão.
Utilizador	O utilizador tem acesso à aplicação móvel desenvolvida e consulta a ocupação estimada das salas e informações sensoriais sobre as mesmas.
Eclipse Hono	Abstracção aos diferentes protocolos dos sensores, que providencia uma forma uniformizada de comunicação com os mesmos.

Tabela 3.1: Atores do sistema

3.1.3 Diagramas de casos de uso

Os diagramas de casos de uso apresentados encontram-se divididos em pacotes: no diagrama 3.1 podem-se visualizar os casos de uso relativos à plataforma de monitorização dos dados sensoriais e aplicação móvel; no diagrama 3.2 tem-se o pacote relativo à plataforma de gestão; por fim, no diagrama 3.3 é possível visualizar os casos de uso relativos à API. De notar, que todos os diagramas possuem uma tabela com uma breve descrição dos casos de uso apresentados.

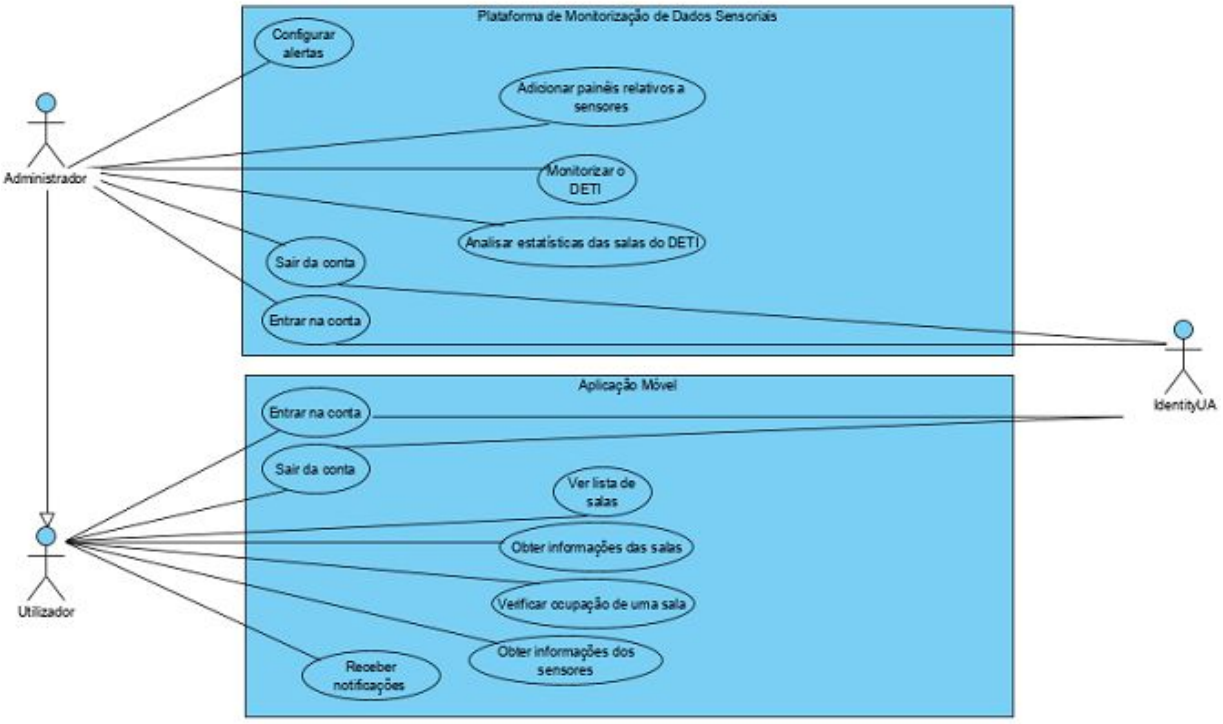

Figura 3.1: Diagrama de casos de uso da aplicação móvel e da plataforma de monitorização

ID	Caso de utilização	Sinopse
1.1	Entrar na conta	Ao abrir a Aplicação Móvel, o Utilizador deverá preencher o formulário corretamente com as suas credenciais de maneira a visualizar com sucesso a aplicação.
1.2	Sair da conta	Após ser autenticado com sucesso na aplicação, o Utilizador possui a opção de sair da aplicação, terminando assim a sessão atual.
1.3	Ver lista de salas	Após ser autenticado com sucesso na plataforma, o Utilizador visualiza uma lista que contém as salas presentes no sistema, a que ele tem acesso.
1.4	Obter informações das salas	Ao receber as salas listadas, o Utilizador pode selecionar uma e receber as informações dos sensores a que tem acesso na mesma, assim como a descrição das salas.
1.5	Verificar ocupação de uma sala	Dentro da lista de salas devidamente equipadas, o Utilizador visualiza uma estimativa do número de indivíduos presentes nas mesmas.
1.6	Obter informações dos sensores	Ao clicar numa sala, o utilizador tem acesso aos dados dos sensores a que ele acesso, assim com informações como o ganho relativamente à medição anterior, um gráfico do último minuto, e um indicador do valor apresentado.
1.7	Receber notificações	Quando algum valor produzido por um sensor se encontrar fora do normal, é enviada uma notificação ao Utilizador com a informação respectiva, caso este escolha receber notificações relativas ao sensor em questão.
2.1	Entrar na Conta	Ao abrir a plataforma, o Utilizador deverá preencher o formulário corretamente com as suas credenciais de maneira a entrar com sucesso na plataforma.
2.2	Sair da Conta	Após ser autenticado com sucesso na plataforma, o Utilizador possui a opção de terminar a sua sessão.
2.3	Configurar alertas	O Administrador pode configurar uma ou mais regras que acionam um alerta relativo a um painel.
2.4	Analisar estatísticas das salas do DETI	O Administrador pode procurar analisar gráficos, com os dados recebidos dos sensores, e mesmo alguns valores indiretos como a média.
2.5	Monitorizar o DETI	O Administrador pode ver os vários valores num mapa, como uma legenda de cores, com os valores mais recentes.
2.6	Adicionar painéis relativos a sensores	O Administrador pode facilmente adicionar um painel com os dados de um sensor existente no sistema.

Tabela 3.2: Descrição dos casos de uso da aplicação móvel e da plataforma de monitorização

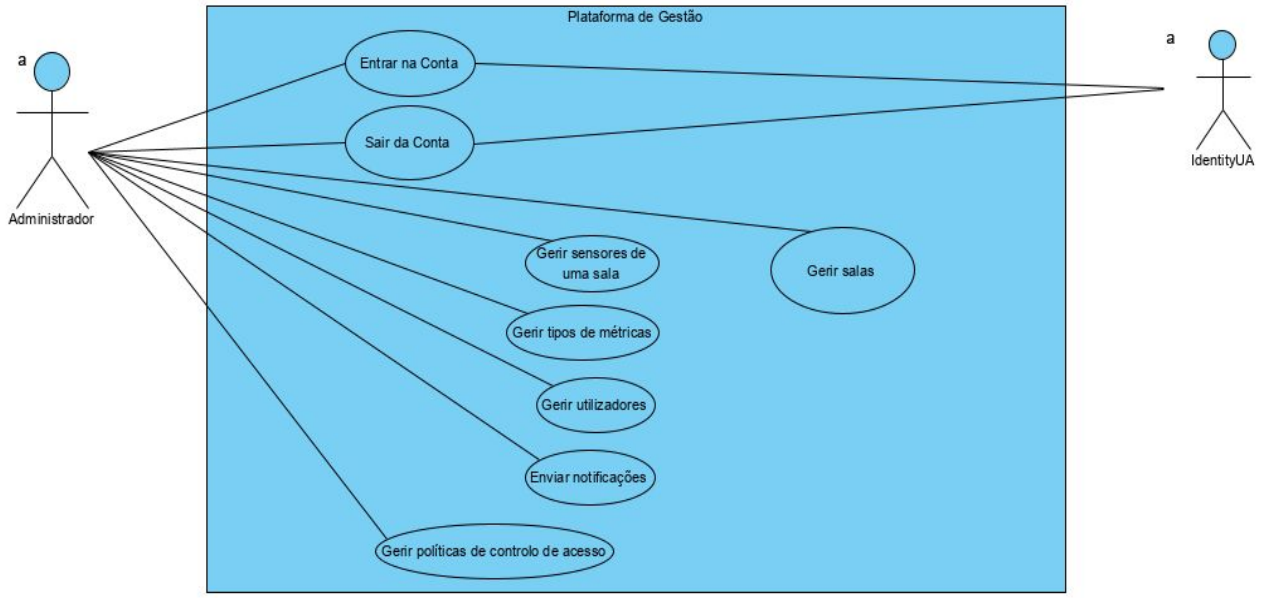

Figura 3.2: Diagrama de casos de uso para a plataforma de gestão

ID	Caso de utilização	Sinopse
3.1	Entrar na conta	Ao entrar na plataforma de gestão, o Administrador deverá preencher o formulário corretamente com as suas credenciais de maneira a entrar com sucesso na plataforma.
3.2	Sair da conta	Após ser autenticado com sucesso na plataforma de gestão, o Administrador possui a opção de sair da plataforma, terminando assim a sessão atual.
3.3	Gerir salas	O Administrador pode aceder à lista de salas do sistema, assim como adicionar, remover ou editar uma sala.
3.4	Gerir sensores de uma sala	O Administrador pode aceder à lista de sensores existentes numa sala do sistema, assim como adicionar, remover ou editar sensores.
3.5	Gerir tipos de métricas	O Administrador pode aceder à lista de tipos de métricas existentes no sistema, assim como adicionar, remover ou editar um tipo de métrica.
3.6	Gerir utilizadores	O Administrador pode aceder à lista de utilizadores do sistema, assim como conferir, adicionar ou remover o estatuto de administrador do sistema.
3.7	Enviar notificações	O Administrador pode enviar notificações para a aplicação móvel, podendo escolher o tópico (controlo ou relativo a um sensor) de envio.
3.8	Gerir políticas de controlo de acesso	O Administrador pode aceder à lista de políticas configuradas para um recurso do sistema, assim como adicionar, editar ou remover políticas relativas a um utilizador ou grupos de utilizadores relativas a um recurso do sistema.

Tabela 3.3: Descrição dos casos de uso da plataforma de gestão

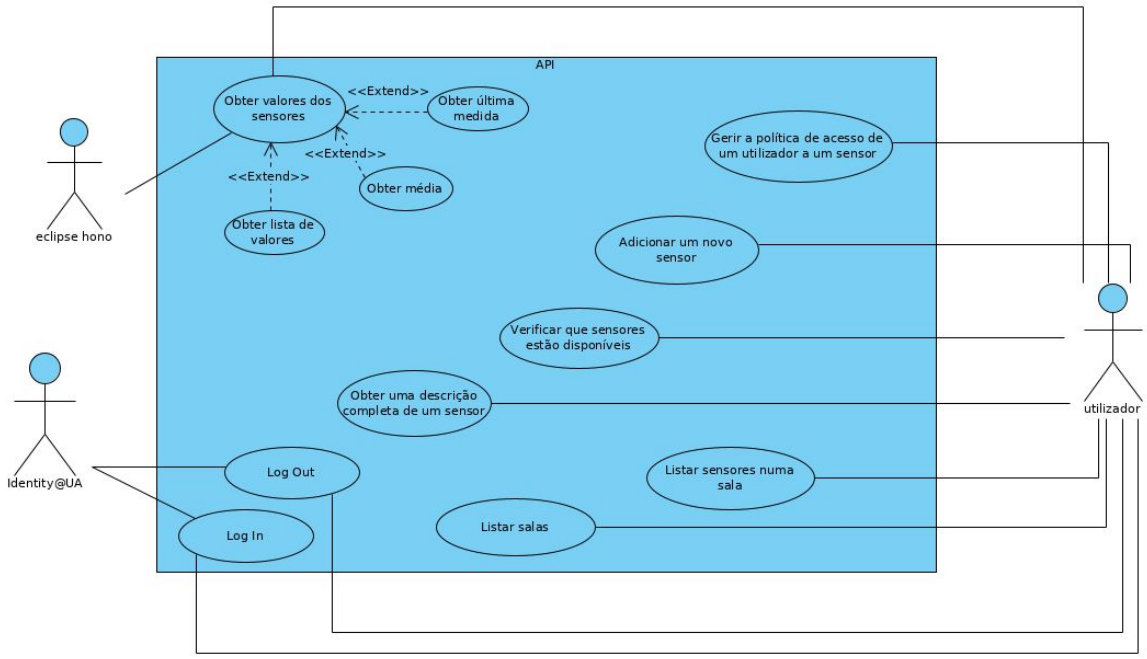

Figura 3.3: Diagrama de Casos de Uso para a API

ID	Caso de utilização	Sinopse
4.1	Obter valores	O Utilizador chama a função da API para obter um dos valores possíveis permitidos pela interface.
4.1.1	Obter última medida	O Utilizador pode optar por fornecer uma sala ou um sensor. No primeiro caso, recebe da API todos os últimos valores medidos nessa divisão, enquanto que no segundo apenas o último valor medido da métrica correspondente ao sensor.
4.1.2	Obter média	O Utilizador fornece um intervalo de tempo sobre o qual a API devolverá as médias das métricas durante o mesmo. Adicionalmente, o Utilizador deve fornecer um sensor de forma a receber somente a média da métrica correspondente ou uma divisão de forma a receber todas as médias das métricas presentes na mesma.
4.1.3	Obter lista de valores	O Utilizador fornece um sensor que será o alvo da API, bem como um intervalo de tempo sobre o qual a API devolverá todos os valores medidos durante o mesmo.
4.2	Verificar que sensores estão disponíveis	O Utilizador verifica a que sensores tem acesso na API.
4.3	Obter uma descrição completa de um sensor	O Utilizador obtém uma descrição de um dado sensor, incluindo o seu tipo, os vários eventos, e o seu ID.
4.4	Listar salas	O Utilizador recebe da API uma lista com as salas equipadas a que tem acesso.
4.5	Listar sensores numa sala	O Utilizador fornece uma divisão e recebe da API uma lista com os sensores equipados, na mesma, a que tem acesso.
4.6	Listar tipos de sensores	O Utilizador recebe da API uma lista com os diversos tipos de sensores existentes.
4.7	Adicionar um novo sensor	O Administrador adiciona um novo sensor à API, com a sua descrição completa
4.8	Gerir a política de acesso de um utilizador a um sensor	O Administrador modifica a regra de decisão de um dado utilizador
4.9	Realizar Log In	O Utilizador realiza um pedido para se autenticar na API
4.10	Realizar Log Out	O Utilizador realiza um pedido à API para se desconectar

Tabela 3.4: Descrição dos casos de uso da API

3.1.4 Requisitos Funcionais

3.1.4.1 Requisitos dos Módulos de Sensorização

Referência	Requisito funcional
RFS-1	Deve ser possível medir a temperatura dentro de uma sala.
RFS-2	Deve ser possível medir a humidade dentro de uma sala.
RFS-3	Deve ser possível medir a pressão dentro de uma sala.
RFS-4	Deve ser possível medir a qualidade do ar dentro de uma sala (Concentração de gases VOC).
RFS-5	Deve ser possível medir a luminosidade de uma sala
RFS-6	Deve ser possível medir o ruído dentro de uma sala.
RFS-7	Deve ser possível medir a concentração de Dióxido de Carbono dentro de uma sala.
RFS-8	Deve ser possível contabilizar o número de dispositivos bluetooth ligados dentro de uma sala.
RFS-9	Deve ser possível medir a corrente elétrica utilizada dentro de uma sala.
RFS-10	Deve ser possível enviar os dados sensoriais por rede.

Tabela 3.5: Requisitos funcionais dos módulos de sensorização.

3.1.4.2 Requisitos da API

Referência	Requisito funcional
RFA-1	Deve ser possível o controlo granular de acessos à informação dos diversos sensores.
RFA-2	Deve ser possível a obtenção de valores dos sensores (Último valor, média, lista exaustiva de medições)
RFA-3	Deve ser possível a obtenção de informação sobre eventos ocorridos (Último evento, contagem, lista exaustiva)
RFA-4	Deve ser possível listar os tipos de sensores suportados no sistema e num dado espaço
RFA-5	Deve ser possível listar os sensores acessíveis a um determinado utilizador
RFA-6	Deve ser possível listar os sensores existentes num dado espaço e os espaços registados no sistema.

Tabela 3.6: Requisitos funcionais da API

3.1.4.3 Requisitos do Frontend

Referência	Requisito funcional
RFF-1	Devem ser listadas as salas que contém os equipamentos de sensorização.
RFF-2	Deve ser possível verificar a ocupação de uma sala.
RFF-3	Devem ser possível obter informações detalhadas de uma sala.
RFF-4	Devem ser disponibilizados dados recolhidos anteriormente, durante um período de tempo.
RFF-5	Deve ser possível alterar a política de utilização de um dado recurso para um utilizador ou grupos de utilizadores.
RFF-6	Deve ser possível a autenticação do utilizador através do IdP.
RFF-7	Deverão ser enviadas notificações ao utilizador, consoante as condições das salas.
RFF-8	Devem ser apresentadas estatísticas das informações obtidas pelos sensores.
RFF-9	Deve ser listados os tipos de sensores e os sensores existentes numa sala.
RFF-10	Deve ser possível gerir os sensores e ver os seus detalhes.

Tabela 3.7: Requisitos funcionais do Frontend

3.1.5 Requisitos Não-Funcionais

3.1.5.1 Requisitos de usabilidade

Referência	Requisito de usabilidade
RU-1	Disponibilizar uma API simples e completa de forma a que futuros projetos possam utilizar os sensores instalados, tirando proveito das suas funcionalidades.
RU-2	A aplicação móvel deverá utilizar os layouts nativos de cada sistema operativo para maximizar a estabilidade e a familiaridade do utilizador com a forma de interação.
RU-3	A solução deverá ser modular de forma a permitir que futuras adições sejam céleres, tanto a nível dos sensores, como a nível da API.
RU-4	Os endpoints disponibilizados pela API devem ser o mais genéricos possível e devem poder ser facilmente identificáveis.

Tabela 3.8: Requisitos de Usabilidade

3.1.5.2 Requisitos de desempenho

Referência	Requisito de desempenho
RD-1	Os sensores de temperatura, pressão e humidade devem disponibilizar novos dados a cada 3 segundos.
RD-2	O sensor de gases VOC deve disponibilizar novos dados a cada 5 minutos.
RD-3	O sensor de luminosidade deve disponibilizar novos dados a cada 1 segundo.
RD-4	O sensor de CO2 deve disponibilizar novos dados a cada 10 minutos.
RD-5	A aplicação móvel necessita de suportar todos os dispositivos móveis com, no mínimo, o sistema operativo Android 5.0.

Tabela 3.9: Requisitos de Desempenho

3.1.5.3 Requisitos de segurança e integridade dos dados

Referência	Requisito de segurança e integridade dos dados
RS-1	O Sistema deverá controlar o acesso aos dados dos sensores consoante as políticas aplicadas a cada utilizador.
RS-2	Apenas os administradores do sistema deverão poder alterar as políticas de acesso utilizadores.
RS-3	Apenas os administradores do sistema deverão poder adicionar novos sensores e seus tipos.
RS-5	Todos as funcionalidades do Sistema estarão disponíveis apenas se o utilizador tiver sido autenticado previamente pelo IdP.
RS-6	Os dados recolhidos pelos sensores são encriptados no envio para o broker.

Tabela 3.10: Requisitos de Segurança e Integridade dos Dados

3.1.5.4 Requisitos de documentação

Referência	Requisito de documentação
RDoc-1	A API deverá possui uma documentação clara, abrangendo os vários serviços.
RDoc-2	O módulo de sensorização deve ser documentado, com um esquema, especificações e descrição dos sensores e módulos individuais integrados.
RDoc-3	As instalações nas diversas salas devem ser documentadas com os procedimentos efectuados e com uma planta/mapa da sala com o módulo instalado.
RDoc-4	Ambas as aplicações deverão possuir uma descrição das várias funcionalidades, e o conjunto de passos para o utilizador usufruir da mesma.

Tabela 3.11: Requisitos de Documentação

3.2 Arquitetura do Sistema

Nesta secção vai ser explorada a arquitetura do sistema, sendo apresentado nas seções seguintes o modelo de domínio, assim como o modelo de tecnologias.

3.2.1 Modelo de Domínio

Relativamente ao modelo de domínio, este descreve como os recursos do sistema estão relacionados entre si, tornando-se útil para se perceber como a informação deve ser armazenada no fluxo de execução do sistema.

Analisando o diagrama, tendo como foco o recurso “Sensor”, verifica-se que cada sensor, como mapeia um dispositivo físico, tem de estar necessariamente num “Espaço”, mesmo que não esteja a ser utilizado. Para além disso, é necessário que cada sensor tenha ter um tipo de métrica de dados, para que as medições provenientes destes façam sentido. De notar que cada “Sala” e cada “Tipo de Sensor” podem ter vários sensores atribuídos.

Posteriormente, cada Utilizador mediado por uma ou mais Políticas de Acesso pode aceder a um conjunto de recursos, como por exemplo os detalhes de uma determinada sala, ou as últimas medições de uma dado sensor.

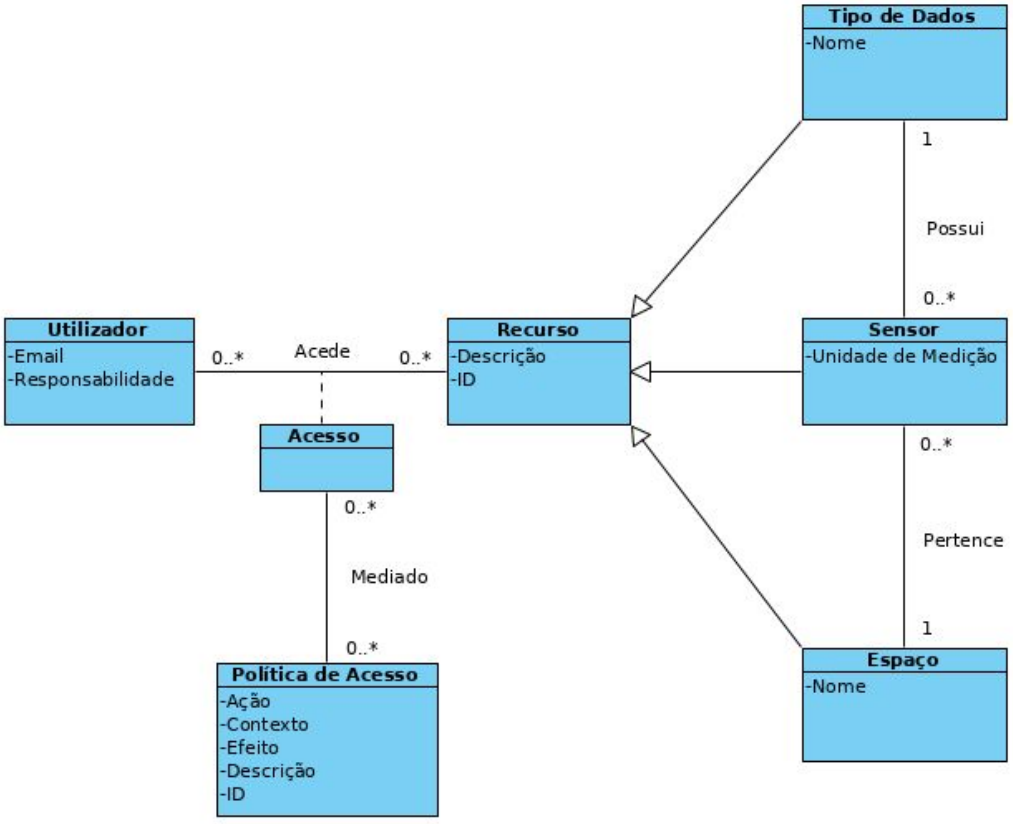

Figura 3.4: Diagrama do modelo de domínio do sistema.

3.2.2 Modelo de Tecnologias

No que toca ao modelo de tecnologias deste sistema:

- Sensores ligam-se a um **Pycom Lopy 1** que formam um módulo localizado num espaço no DETI, denominado de ISU (Integrated Sensorization Module).
- O Lopy envia os dados sensoriais, por MQTT, para um **Raspberry Pi**, que funciona como uma ponte MQTT (denominado de forma mais comum por bridge MQTT), que liga posteriormente ao broker agregador de dados de sensorização de todos os módulos.
- Os dados enviados para esse broker são persistidos numa base de dados time-series InfluxDB.
- A API persiste os dados de atributos de salas, sensores, e dados básicos do utilizador
- O acesso posterior a recursos e dados é controlado por regras internas e um módulo de **controlo de acessos baseado em atributos** (ABAC), que define e avalia políticas utilizando um conjunto de atributos, persistidas numa base de dados orientado a documentos, **MongoDB**.
- Todos os recursos são **geridos por uma plataforma** web de gestão
- A exposição de recursos e dados sensoriais é **visualizada** sob a forma de aplicações web ou mobile.

Abaixo, é possível visualizar o diagrama geral da arquitetura de tecnologias do sistema.

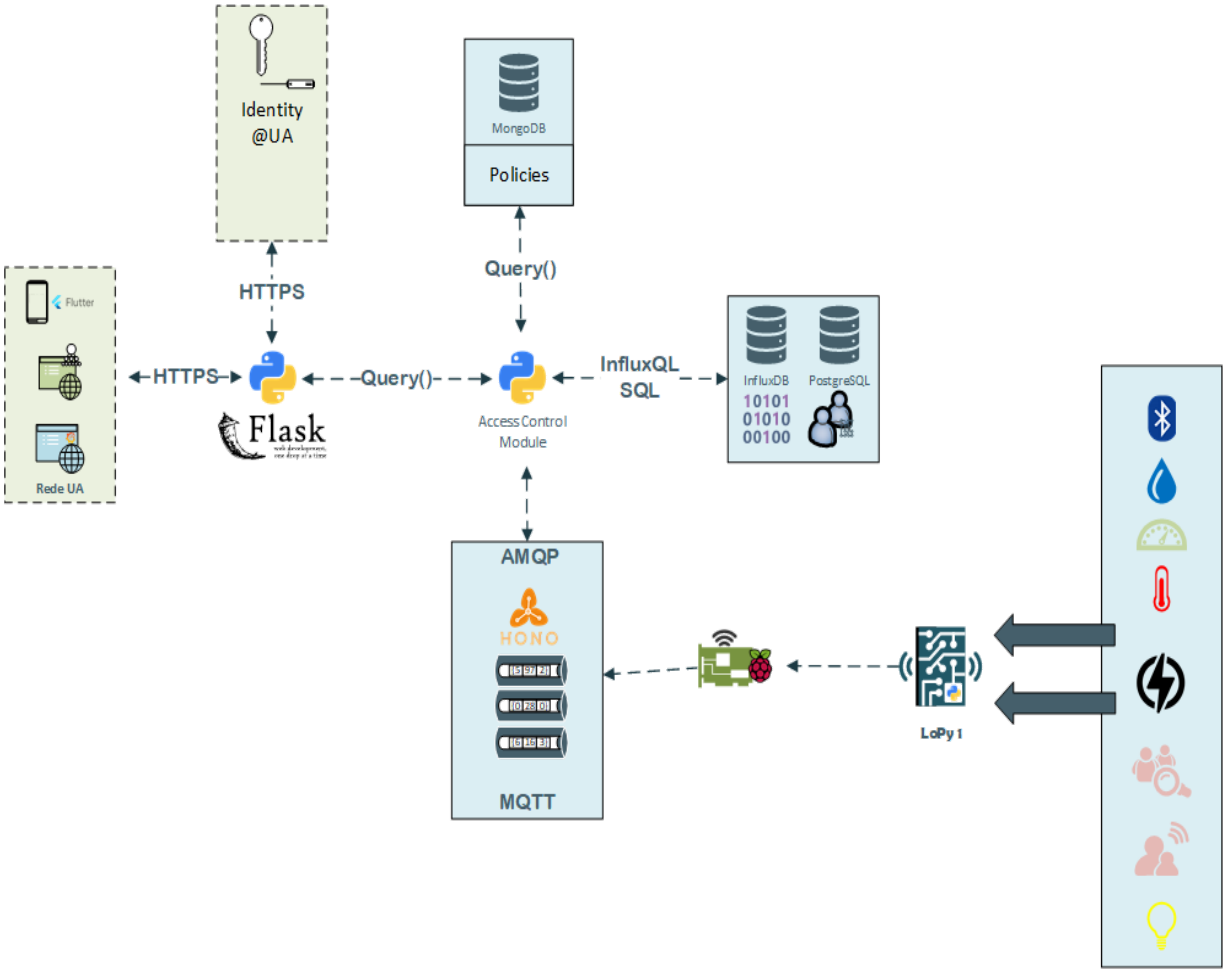

Figura 3.5: Diagrama do modelo de tecnologias do sistema.

3.2.3 Modelo de Instalação

No que toca ao modelo de instalação, o seu desenho foi pensado para poder suportar um sistema de arquitetura em micro-serviços. O diagrama abaixo permite verificar que o módulo da API é um ponto central na funcionalidade do sistema, na qual praticamente todos os módulos integram ou mediam recursos que a API expõe.

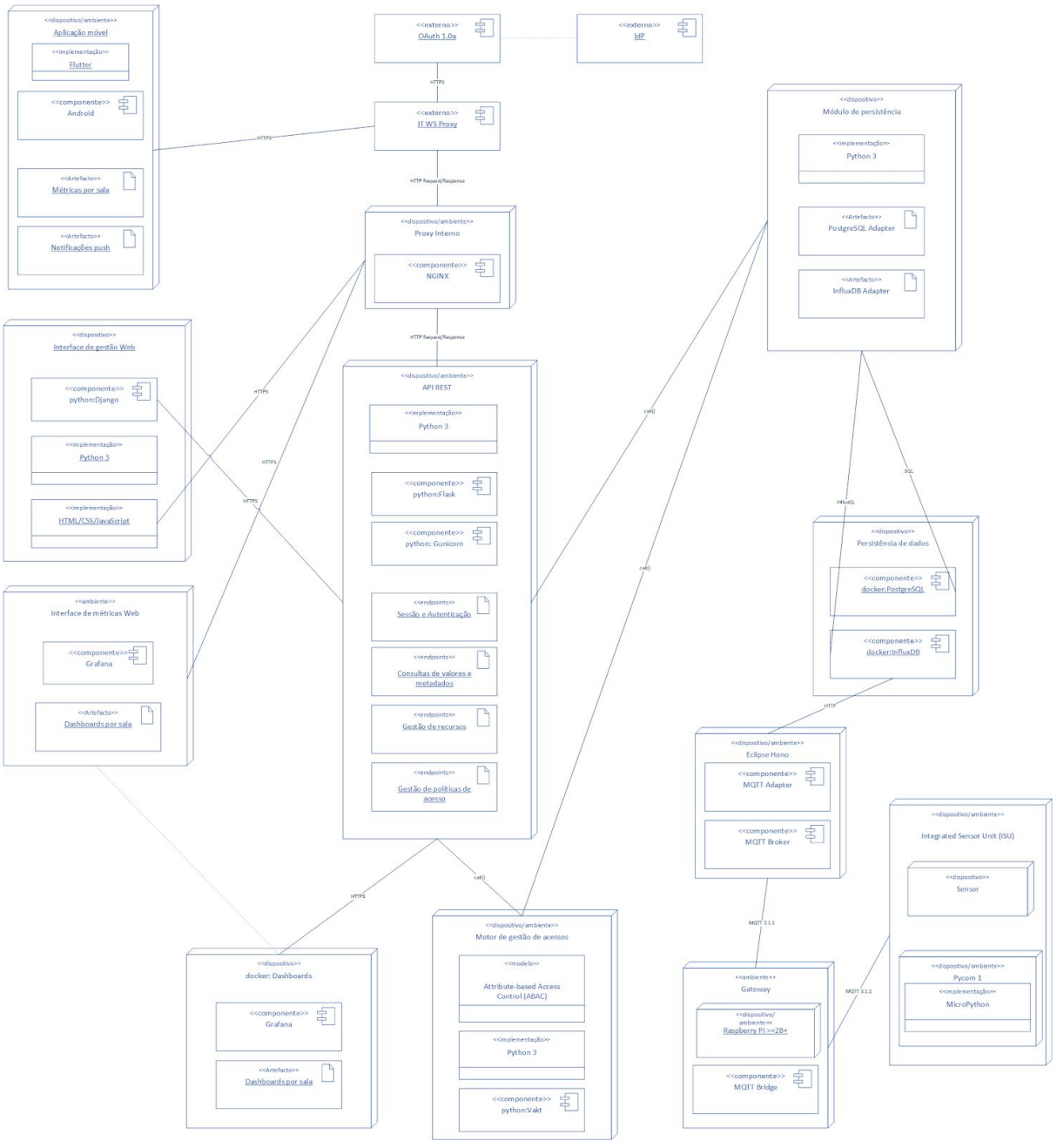

Figura 3.6: Diagrama do modelo de instalação do sistema.

4 Implementação

4.1 Repositórios de Código-Fonte

Todos os repositórios utilizados no projeto foram criados sob a organização DETImotica na plataforma *Github*.

O desenvolvimento foi realizado, onde possível, no branch *develop* até se alcançarem as condições de estabilidade desejadas, procedendo-se então ao *merge* para o branch *master*.

Repositório	Função
detimotica.github.io	Website e Documentação do projeto
API	Código da API e módulos de autenticação, permissões, e interação com as bases de dados
ISU	Código dos módulos de sensorização (Obtenção de valores e envio dos mesmos para o broker MQTT local)
SensorGateway	Código e Configuração da gateway MQTT (Receção dos dados sensoriais e envio para o Eclipse Hono)
MobileApp	Aplicação Mobile de demonstração da API (Consulta da ocupação das salas e de dados sensoriais)
PlataformaGestao	Plataforma Web de Gestão de sensores, salas, utilizadores, e políticas de acesso

Tabela 4.1 : Listagem de links e funções dos vários repositórios do projeto

4.2 Sensores

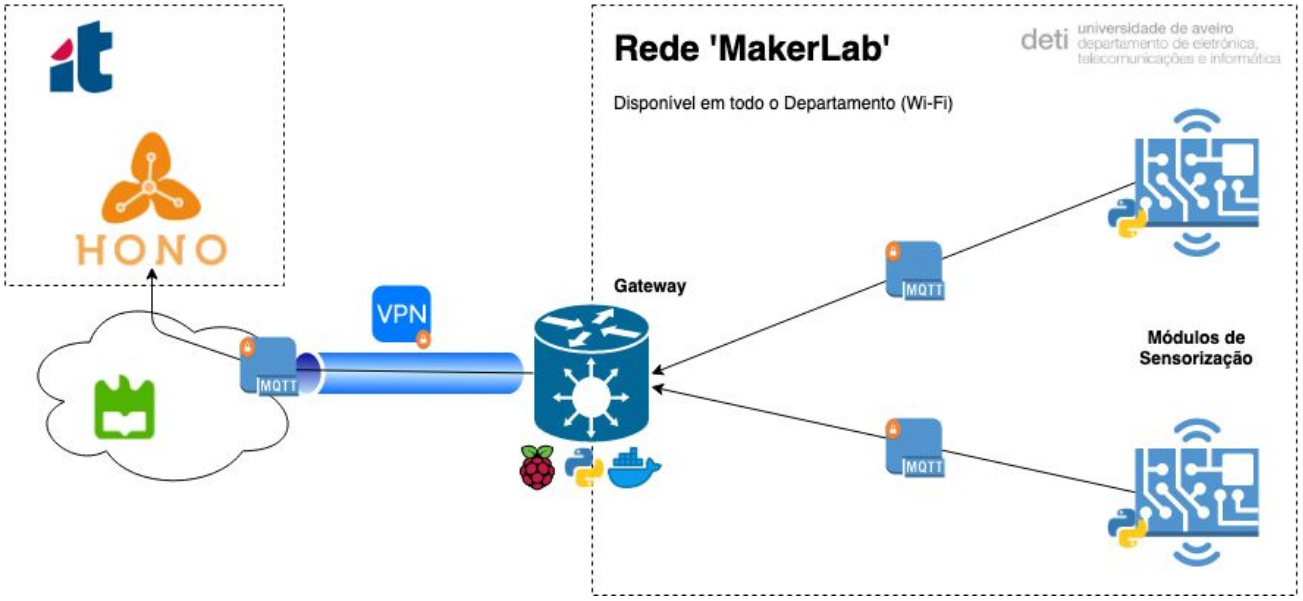

Figura 4.1 : Vista Geral da Arquitetura

4.2.1 Módulos sensoriais

Microcontrolador

Dadas as suas capacidades de input/output, comutação, e memória e também o facto de existirem unidades disponíveis para requisição no armazém de componentes do DETI MakerLab, o microcontrolador escolhido para o desenvolvimento deste projeto foi o **LoPy versão 1**:

- Baseado no Espressif ESP-32
- Dispõe de Wi-Fi e Bluetooth integrados
- Programável em Micro-Python
- Suporta aplicações multi-threaded
- Possui vários ADCs e I2C Bus

A release de firmware utilizada foi a '1.18.2', com a versão '*v1.8.6-849-07a52e4*'.

No entanto, estudantes e docentes do departamento poderão utilizar qualquer microcontrolador para efetuar instalações em novas salas no futuro,

bastando para isso seguirem a especificação contida nas secções seguintes deste documento.

Sensores instalados

Em cada montagem foram instalados os seguintes sensores:

Sensor	Métrica	Precisão	Comunicação	Alimentação
LMV324 (SEN-12642)	Som (dB)	---	Medições Analógicas	5V
TSL2561	Luminosidade (lux)	---	I2C	3.3V
BME860	Temperatura (ºC)	+- 1ºC	I2C	3.3V
	Humidade (%RH)	+- 3%RH		
	Pressão (hPa)	+- 1hPa		
	Indoor Air Quality (Índice - Gases VOC)	---		

Tabela 4.2 : Listagem de sensores requisitados e instalados

Adicionalmente, é registado ainda o número de dispositivos BLE visíveis dentro do alcance do módulo Bluetooth integrado no LoPy.

Durante o desenvolvimento foi também instalado e programado com sucesso um sensor de temperatura e humidade DHT-11.

Foi igualmente preparado código-fonte para um sensor de concentração de gás Dióxido de Carbono MG812, que não foi possível instalar.

No entanto, a plataforma é dinâmica e não restringe os tipos de sensores que podem ser utilizados, pelo que cada montagem futura poderá possuir um conjunto distinto de sensores.

Montagem

As duas montagens exemplo preparadas ao longo do desenvolvimento do projeto foram realizadas de acordo com o seguinte diagrama:

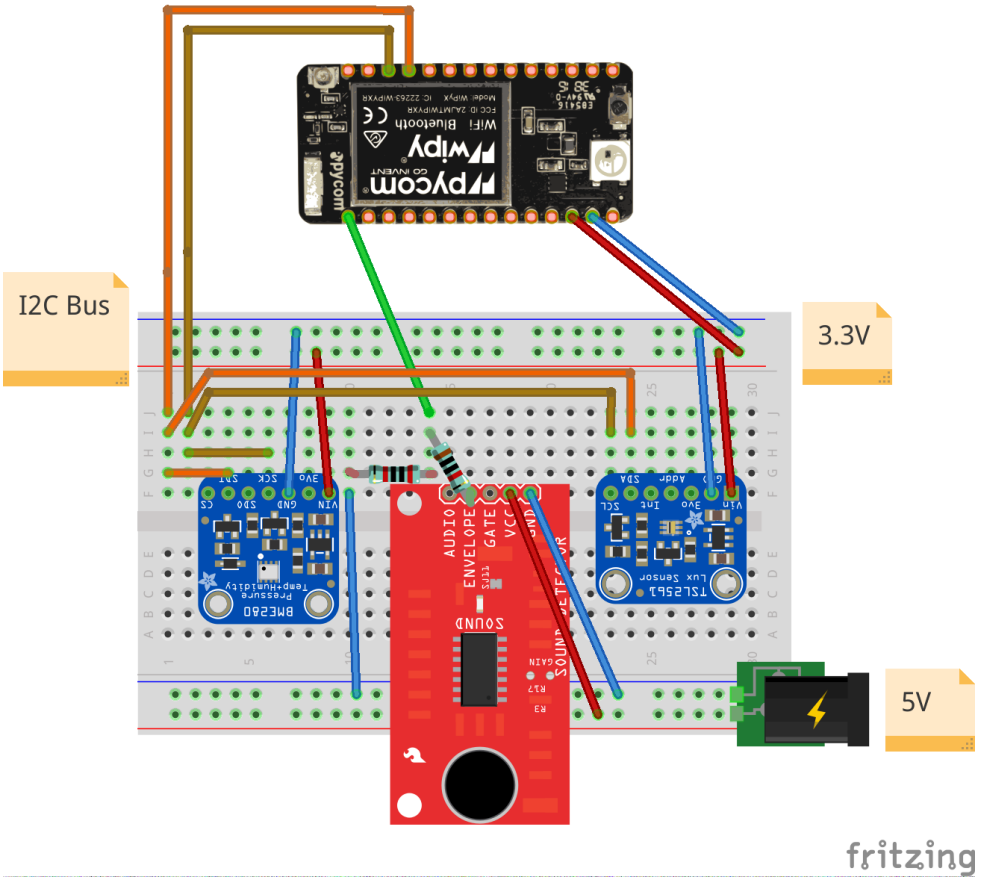

Figura 4.2 : Diagrama de montagem dos módulos de sensorização integrados

De notar a necessidade de utilização de uma pequena placa de alimentação externa para o(s) sensor(es) que necessitem de input de 5V não fornecido pelo microcontrolador. Por conseguinte, é importante que os sinais de output desses sensores sejam sujeitos a um divisor de tensão de forma a limitar a voltagem a 3.3V e proteger o LoPy.

4.2.2 Configuração e Utilização

O código-fonte de um módulo sensorial é dividido em duas partes lógicas:

- Inicialização, conexão, e envio de mensagens
- Lógica inerente a cada sensor individual

A listagem e configuração dos sensores é realizada no ficheiro auxiliar '*conf.json*'.

Cada sensor instalado poderá disponibilizar várias métricas ao sistema, sendo que todas necessitam de ser registadas através de pedido à equipa de administração (que utiliza a plataforma de gestão para esse efeito). Após o registo é fornecido, para cada métrica, um par '*ID / Chave de encriptação*' que deverá ser introduzido no ficheiro supracitado conforme o exemplo seguinte.

```
{
    "isu_id": "cc2f8ad2-e719-4366-a157-5463c9097dfc", // UUID do módulo
    "modules": [
    {
      "name": "tsl2561",
      "active": true,
      "wait_time": 6000, // em milisegundos
      "metrics": {
        "lux": {         // nome arbitrário - denomina uma métrica
          "id": "bb60686d-614a-43af-bf51-cc41477326d9", // UUID da métrica
          "key": "ck5PZQLoy0DleIYQkoRvkw==" // Chave AES-128
        },
       // ...
      }
    },
    // ...
}
```

Figura 4.3 : Exemplo de configuração do ficheiro conf.json

Após ter adicionado um novo sensor ao ficheiro de configuração, a programação da lógica de obtenção de medições do mesmo é feita através da adição de um novo ficheiro python no diretório *sensors* com o nome configurado no campo *name*.

Seguindo a configuração acima como exemplo, o código associado ao sensor é colocado em '*sensors/tsl2561.py*'.

Durante o seu processo de inicialização, o microcontrolador tenta importar todos os ficheiros associados a sensores que constem no ficheiro de configuração e executa uma vez o método *setup* de cada um deles.

Posteriormente, o método *loop* será invocado repetidamente com o intervalo configurado em *wait_time*.

```
from lib.tsl2561_driver import *
lux_sensor = None

def setup(dm):
   global lux_sensor
   lux_sensor = device()
   lux_sensor.init()

def loop(dm): # Recebe 'dm' como argumento para poder publicar
telemetria
   value = lux_sensor.getLux()
   print("Lux value: " + str(value))
   dm.publish("lux", value) # nome 'lux' é traduzido pelo ID no
conf.json
```

Figura 4.4 : Exemplo de programação da lógica de obtenção de dados de um sensor

A telemetria poderá ser publicada com o auxílio do objeto *detimotic* (*dm*) fornecido como argumento às funções. Para tal basta indicar o valor a enviar e o nome da métrica correspondente, já que um sensor pode dispor de várias métricas associadas. O nome é depois traduzido automaticamente pelo ID equivalente.

Resumindo, as acções necessárias para a adição de um novo sensor à lógica de execução do microcontrolador são apenas:

- Configuração do mesmo no ficheiro *conf.json*
- Adição de um ficheiro *sensors/<name>.py* com a interface demonstrada no exemplo

De forma a assegurar um elevado nível de parametrização, o ficheiro *detimotic_conf.json* (exemplificado em baixo) permite a configuração de várias facetas da lógica de conexão e envio de informação.

```
{
  "wifi": {
      "ssid": "MakerLab",
      "passw": "p4ssw0rd"
  },
  "gateway": {
      "addr": "192.168.0.2",
      "uname": "detimotic",
      "passw": "mqtt_password",
      "port": 1883,
      "telemetry_topic": "telemetry",
      "ping_freq": 10000 // em milisegundos
  },
  "watchdog": 5000 // em milisegundos
}
```

Figura 4.5 : Exemplo de configuração do ficheiro detimotic/detimotic_conf.json

Para assegurar a recuperação do funcionamento do módulo no caso de um erro imprevisto que seja fatal à lógica do mesmo, é utilizado um *watchdog timer* que o reinicia e cujo limite é igualmente configurável.

4.2.3 Segurança nas comunicações

De modo a garantir a segurança nas comunicações MQTT com dados de telemetria dos microcontroladores para a gateway, a payload das mensagens é encriptada utilizando AES-128 e de acordo com o esquema em baixo.
A escolha de algoritmo visou limitar o impacto na performance dos microcontroladores sem comprometer a segurança do sistema.

Esta solução impede que terceiros consigam visualizar informação sensorial à qual poderiam não ter acesso normalmente.

Adicionalmente, o facto de cada sensor possuir a sua chave de encriptação única impossibilita o envio de dados sem o conhecimento da mesma, pelo que impede terceiros de 'se fazerem passar' por um sensor existente e enviar dados falsos.

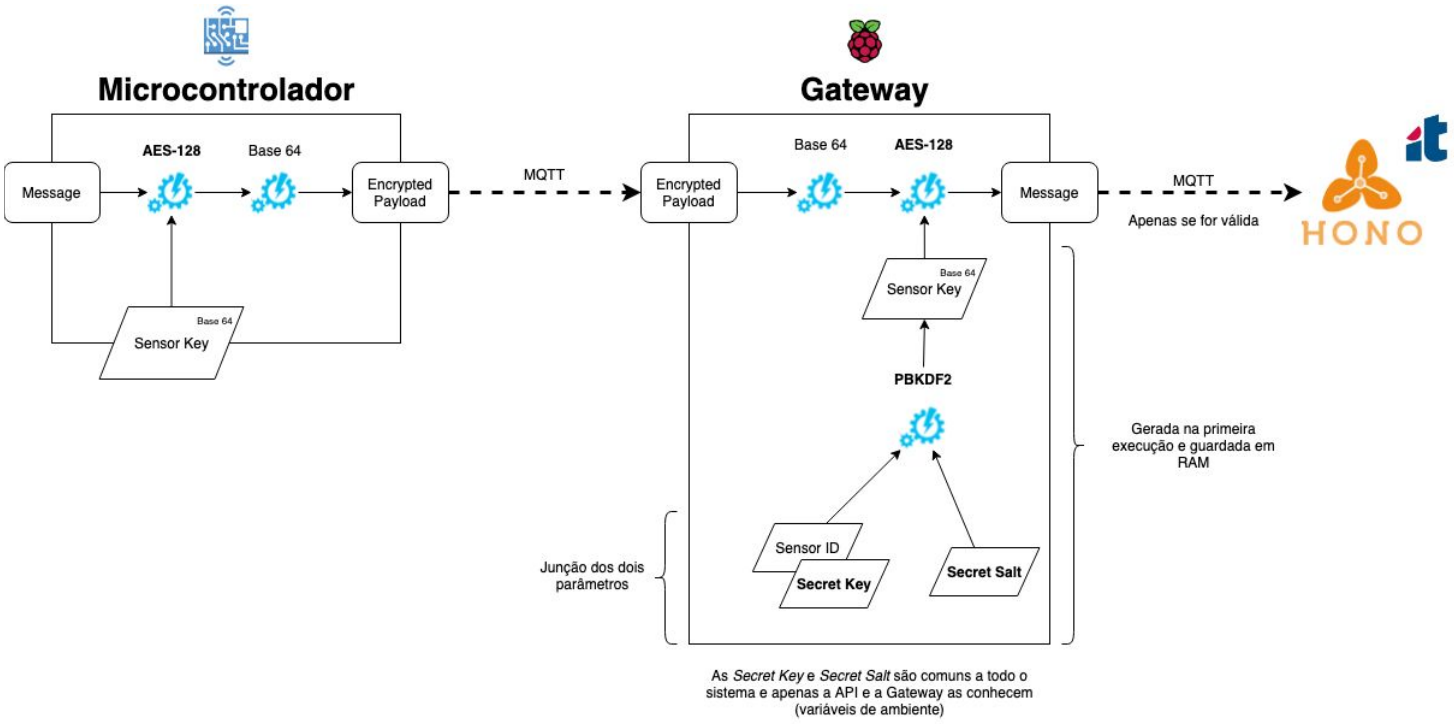

Figura 4.6 : Diagrama ilustrativo da arquitetura de segurança nas comunicações

De modo a evitar a manutenção de uma lista de chaves de encriptação na gateway, a chave de cada sensor é dedutível através do seu UUID e de um conjunto de chaves secretas que apenas a API e a gateway conhecem.

4.2.4 Gateway

Equipamento agregador que está conectado à rede do MakerLab e recebe todas as mensagens MQTT enviadas pelos módulos de sensorização.

Dispõe também de uma ligação à rede da Universidade de Aveiro através da VPN da mesma, por onde os dados recebidos são encaminhados para o broker da instância Eclipse Hono alojada no Instituto de Telecomunicações.

Assim, de um ponto de vista de alto-nível, o equipamento efetua as funções de uma bridge MQTT.

Durante o desenvolvimento do projeto foi utilizado um Raspberry Pi (2B ou 4B) para esse efeito.

4.2.5 Broker MQTT

O broker local para o qual toda a telemetria é enviada pelos microcontroladores pode estar alojado em qualquer servidor acessível dentro da rede do MakerLab.

Durante o desenvolvimento do projeto o local escolhido foi a própria gateway, sendo que o software selecionado foi o *Mosquitto* (imagem Docker: eclipse-mosquitto)

As configurações utilizadas podem ser encontradas no repositório respetivo.

4.2.6 Receção, desencriptação, e encaminhamento de mensagens

A lógica de receção, desencriptação, e encaminhamento de mensagens é assegurada por um programa multi-threaded em *Python* (*gateway.py*).

A thread principal conecta-se ao broker MQTT local e subscreve o tópico '*telemetry*' e todos os seus subtópicos (i.e. '*telemetry/#*').

As mensagens dos microcontroladores com valores de um determinada métrica deverão ser enviadas para o tópico: 'telemetry/<ID do sensor/métrica>' e, conforme o exposto na secção anterior, ter sido encriptadas com a sua chave única AES-128 e posteriormente codificadas em base64.

Exemplo:

```
telemetry/7d245a97-66c7-49eb-9940-dbb9cb24f5ec HLhUwpmNgorg7W6lf4WZhcSXMThCJ8wHcv+wavM=
```

Para o processamento das mensagens é utilizado um modelo *producer/consumer* com múltiplas threads.

No momento de receção, a mensagem é mantida numa fila até uma das *worker threads* se encontrar disponível para a desencriptar, verificar a sua integridade, e proceder ao seu encaminhamento para o broker da instância Eclipse Hono.

A mensagem, uma vez desencriptada, deverá ter o seguinte formato (no caso do LoPy é assegurado automaticamente):

```
{'value': <valor observado pelo sensor>}    // Exemplo: {'value': 1018.56}
```

Existe ainda uma thread adicional para manter o estado da conexão com o broker remoto.

Durante o desenvolvimento do projeto foi utilizado um máximo de 20 *worker threads* concorrentes.

4.2.7 Configuração

De forma a assegurar um elevado nível de parametrização, o ficheiro *gateway_config.json* (exemplificado em baixo) permite a configuração de várias facetas da lógica de receção e encaminhamento.

```
{
      "remote": {   // Configurações do broker remoto
          "host": "iot.av.it.pt",
          "port": 1883,
          "tenant_id": "detimotic",
          "device_prefix": "device_", // prefixo ao UUID da métrica
          "telemetry_topic": "telemetry",
          "value_description": "value"
      },
      "local": {   // Configurações do broker local
          "host": "localhost",
          "port": 1883,
          "uname": "detimotic",
          "telemetry_topic": "telemetry",
          "value_description": "value",
          "max_workers": 20
      },
      "security": {
          "kdf_iterations": 1974  // Iterações do PBKDF2
      }
}
```

Figura 4.7 : Exemplo de configuração do ficheiro gateway_config.json

Existe ainda o ficheiro escondido *.secret_config.json* (exemplificado em baixo) que não deverá ser público e onde são introduzidos os parâmetros de autenticação e segurança.

```
{
      "local_broker_pw": "mqtt_password",
      "hono_sensors_pw" : "pw_that_proves_gw_identity_to_hono",
      "secret_key": "secure_key_for_generating_aes_keys",
      "secret_salt": "random_str123"
}
```

Figura 4.8 : Exemplo de configuração do ficheiro .secret_config.json

4.3 API

4.3.1 Introdução

Relativamente à API, esta surge no sistema como a interface da camada de sensores, possuindo então um conjunto de endpoints desenvolvidos com recurso à framework Flask, que permitem que os programadores e utilizadores finais acedam aos dados dos mesmos, e ainda que possam interagir com esta, por exemplo, criando novas salas, atribuindo sensores a espaços ou alargando o número de ações que dados utilizadores podem realizar **(Subseção 4.3.2).**

Todos os endpoints são autenticados, isto é, apenas utilizadores que pertencem à Universidade de Aveiro são capazes de usufruir do sistema. Para isso, foi feita uma integração de um servidor de autorização, implementando por cima desta serviços de cache de identificação do utilizador e validação das sessões, que permitem também a autenticação do sujeito que concedeu a autorização.

Para além disso, a API está diretamente integrada com o módulo de controlo de acessos, que serve para a gestão das permissões dos utilizadores, por forma a que o acesso aos recursos da API, como Salas e Sensores, possa ser personalizado para cada um por criação de políticas. Este módulo será especificado na **Subseção 4.3.4**.

A API conecta-se também a uma Base de Dados auxiliar para garantir a persistência da meta-informação associada aos dados do sistema, e ainda guardar as relações entre os mesmos (**Subseção 4.3.3)**.

Por fim, a API é configurada por ficheiros para o efeito, e é executada a partir de um Web Server Gateway Interface (WSGI), que a expõe e intermedia todos os pedidos a esta assincronamente, garantindo uma performance significativamente maior do que se fosse executada diretamente. A configuração do WSGI garante, ainda, segurança nas comunicações ao nível dos pedidos.

4.3.2 Endpoints

Desenho da Solução

Especificando os endpoints desta solução, durante a sua definição adotou-se o estilo de desenho de APIs REST: para tal, mapeou-se cada um dos principais recursos e as suas operações num conjunto de endpoints que tem sempre como Base o nome do recurso (Ex: /<nome_recurso>).

Posteriormente, fez-se uso das opções de ação nativas nos *requests* do protocolo HTTP/HTTPS para diferenciar, num mesmo endpoint, o resultado que é esperado da sua execução, deixando deste modo a API simples e intuitiva de utilizar. Resumo nas **Tabelas 4.3 e 4.4** abaixo.

Recurso	**Endpoint Base Path**
Salas	/room
Sensores	/sensor
Tipos de Sensores	/type
Utilizadores	/user
Políticas de Acesso	/accessPolicy

Tabela 4.3 : Resumo dos recursos na API

Ação	**Resultado Esperado**
GET	devolva valores de um recurso
POST	modificação (alteração dos campos do recurso)
DELETE	apague um dado recurso

Tabela 4.4 : Resumo das ações na API

Outras considerações que se tomam em conta neste desenho é o uso do plural para indicar que a ação atua sobre vários recursos ([GET] /rooms), e no caso em que este se encontre no singular, surge um parâmetro adicional para indicar qual o recurso que vai ser acedido, sendo que apenas será omitido no caso do recurso ainda não existir (que é o caso de uma criação).

Exemplos:
[POST] /sensor - criar um novo sensor
[POST] /sensor/<id> - modificar um sensor com o id <id> já existente

Além disto cada recurso pode ainda possuir Opções ou Recursos Internos: nestes dois casos, decidiu-se que deve ser também indicada a opção, após a escolha do recurso ([/ recurso_base] [/ opção/recurso_secundário]).

Exemplos:
[GET] /sensor/<id>/measure/<tipo_medição> - obter os valores de um sensor segundo a forma especificada dentro da opção *measure*
[GET] /room/<id>/sensors - obter os sensores existentes na sala <id>

Para terminar ainda se implementou alguns endpoints diretos, que participam na autenticação dos utilizadores na API. Estes são descritos abaixo.

Sessão autenticada

Todos os pedidos de acesso a recursos pelos endpoints da API são autenticados. Para isto, é necessária a definição de endpoints especiais de sessão que consigam realizar um mecanismo capaz de autenticar o utilizador final. Adicionalmente, a definição de uma API com controlo de acessos por atributos requer que exista recolha de atributos relacionados com o utilizador que, depois, sejam utilizados para definir políticas que controlem finamente os recursos existentes. Com isto, foi utilizado o Identity@UA, que é um servidor OAuth (Open Authorization), versão 1.0a, que integra a autenticação dos utilizadores associados à Universidade de Aveiro pelo IdP (Identity Provider). Isto permite que a API tenha, a partir de um token de acesso, permissão por parte do *end-user* para recolher atributos de identidade.

Por si só, este método não faz qualquer autenticação - para o efeito, uma sessão é criada (a partir de uma cookie de sessão), internamente mapeada para um token de acesso, que fica em cache, e validada sempre que o utilizador fizer um pedido à API.

Para mais informação acerca da implementação do servidor OAuth, por favor, ver a página e a documentação do Identity@UA.

Login do utilizador

O utilizador inicia a sessão chamando o endpoint associado para o efeito (/login). A API faz um pré-processamento para averiguar se o utilizador já está autenticado com sucesso no sistema. No caso de estar, então o processo é finalizado diretamente com a resposta de redirecionamento ou de sucesso final.

Dependendo com que propósito foi chamado, o seu URL poderá ter parâmetros "app" (ID de aplicação) e "redirect_url" (página de redirecionamento após autenticação bem sucedida). Se estes existirem, o primeiro passo consiste em incluí-los na cookie de sessão. Esta fase aplica-se à autenticação dentro da plataforma de gestão e do serviço de dashboards, que diferem do passo final.

Neste ponto, a API começa o processo OAuth, pede o token de pedido e a chave correspondente, e redireciona o utilizador para o IdP da Universidade. De seguida, o utilizador introduz as suas credenciais, valida-as e responde ao consentimento para permissão de utilização de dados pela API, enviado pelo servidor de OAuth. Este, seguidamente, comunica à API pelo endpoint de callback, configurado aquando do registo inicial da API no servidor (/auth_callback).

No processo deste endpoint, é verificado se o utilizador consentiu os dados. Se respondeu negativamente no ecrã de consentimento, o URL que o servidor OAuth envia à API deve conter o parâmetro "*consent=no*". Neste caso, o processo de login aborta e envia uma resposta ao utilizador que o avisa que o processo falhou, informando-o que não consentiu a permissão. No caso de ter aceite o consentimento, é feito um pedido ao access token com o token de pedido e a chave passada no início do processo. O servidor retorna o token de acesso e chave.

Com este token, de seguida, é feito um pedido de todos os atributos do utilizador, seleciona-se e processa-se os que são relevantes para a definição do conjunto de atributos para efeitos de controlo de acesso do utilizador, e guarda-se o resultado em cache de dados local. Os atributos básicos que identificam o utilizador são guardados na cookie da sessão, e internamente mapeados com estas (access token e chave) noutra cache local da sessão.

Ainda falta persistir dados básicos de utilizador na base de dados relacional de atributos para haver um registo de utilizadores que, no passado, utilizaram a plataforma, e conseguir admitir administradores do sistema. Assim sendo, se o utilizador ainda não existir na base de dados, este é inserido com as informações básicas (email e ID), sem o papel de administrador.

Por fim, para concluir o processo de login com sucesso, é necessário responder de acordo com o início do processo:

- Se foi identificado que a autenticação veio da plataforma de gestão, então é necessário passar a string da cookie de sessão pelo URL

como parâmetro. Como a plataforma de gestão apenas se aplica a administradores, isto é verificado. Se sim, faz-se um redirecionamento para o URL que se encontra na sessão, como o valor de “redirect_url”. Se não, à localização é juntado o caminho para uma página de erro “forbidden”.

- Se foi verificado que este início de sessão veio de uma chamada ao endpoint pela entrada numa das páginas que correspondem ao caminho do serviço de dashboards, então é feita uma cookie especial para o efeito, sendo que a cookie de sessão original é a do servidor que expõe esse serviço. Por fim, a resposta redireciona para a página inicial das dashboards.
- Se veio de outro tipo de chamada (direta do endpoint no browser ou na aplicação), então o endpoint responde com sucesso, em que inclui nessa resposta os dados básicos do utilizador.

Abaixo apresenta-se o diagrama de fluxo principal, para compreensão mais rápida.

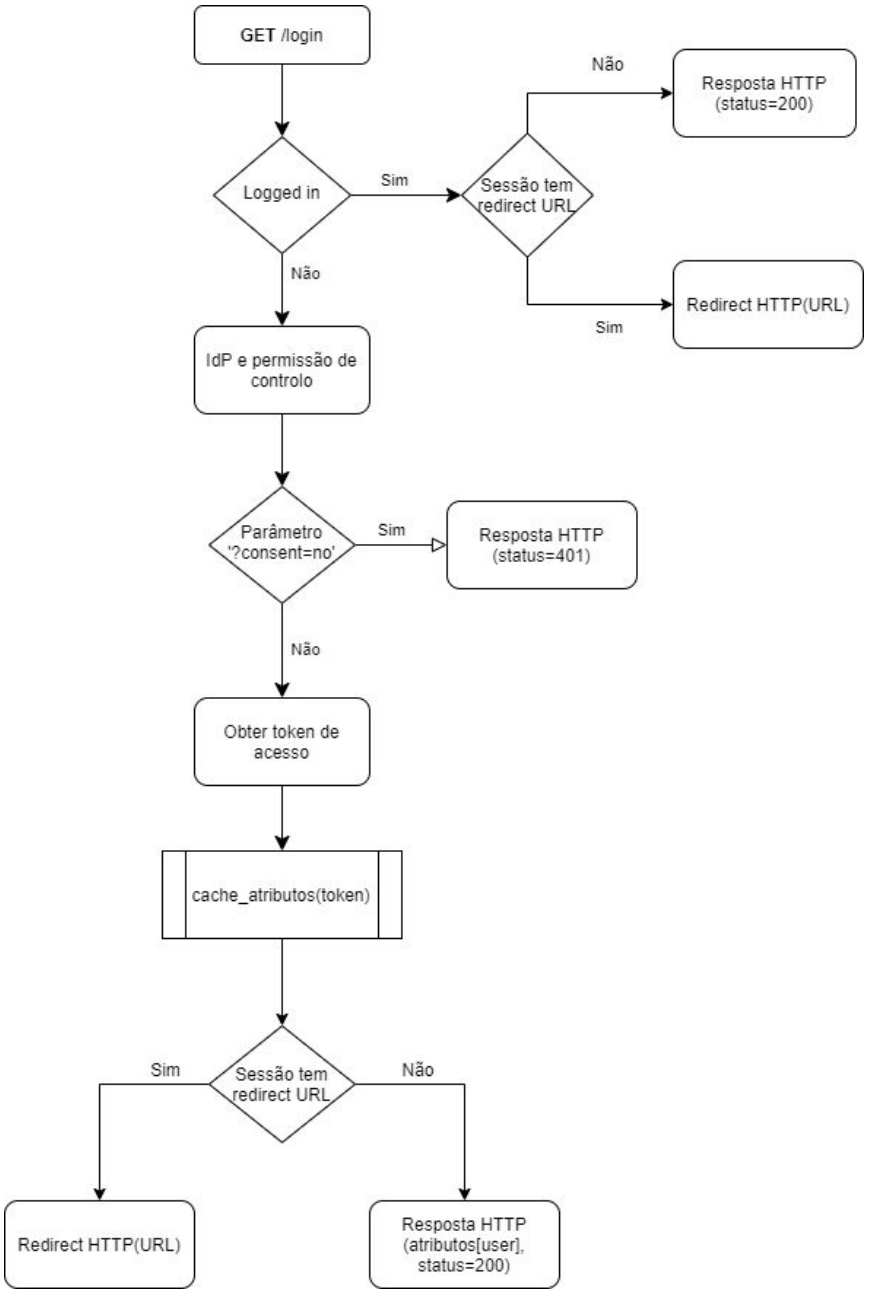

Figura 4.9: Flowchart principal do processo de autenticação.

Logout do utilizador

O processo de logout inicia quando o utilizador final chama o endpoint /logout. Primeiro, é verificado se este não se encontra autenticado. Nesse caso, é informado da situação, e o processo termina. No caso de não estar autenticado, então o processo de desautenticação começa:

- Elimina-se as entradas relacionadas com o utilizador da cache de sessão, expirando assim o token de acesso (do sistema interno).

- Elimina-se da cache de atributos todos os dados relacionados com o indivíduo autenticado
- Limpa-se a sessão
- Se existir uma outra cookie associada ao processo de criação da sessão autenticada, esta é expirada.
- Por fim, é retornada uma resposta de desautenticação bem sucedida.

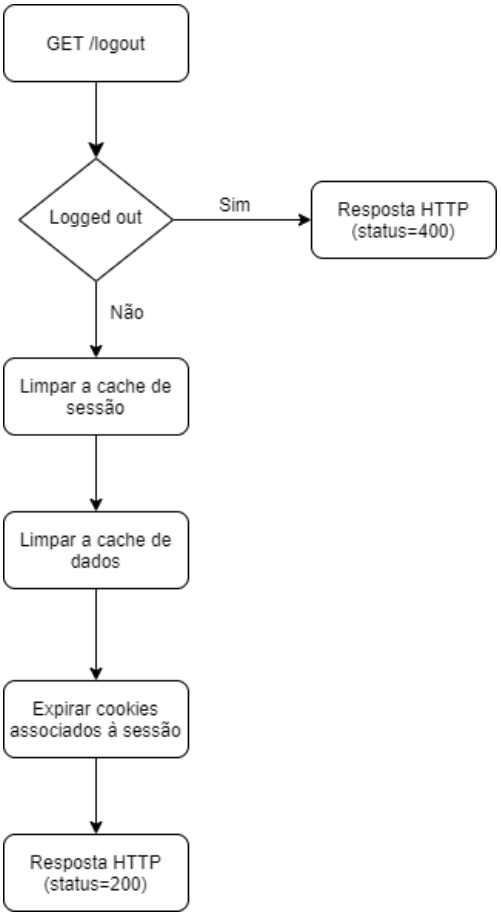

Figura 4.10: Flowchart principal do processo de desautenticação

Sumário das Funcionalidades

Nesta seção, é sucintamente descrito o que cada endpoint é esperado que faça. Para efeitos de simplificação, não se indica que ação do protocolo *http/https* deve ser usada, nem quais são os parâmetros necessários no corpo do pedido, nem quais são as possíveis formas de erros e seus códigos.
Caso seja necessário e precise de procurar saber mais acerca destes detalhes, é possível recorrer à especificação em Swagger (Documentação da API).

Rooms

- **/rooms -** *Devolve todas as salas a que os utilizadores da API têm acesso*
- **/room -** *Serve para um administrador poder criar uma nova sala na API com os detalhes especificados*
- **/room/< ID > -** *Serve para que um utilizador possa gerir/aceder aos detalhes de uma determinada sala com o identificador interno "ID";*
  Ações Possíveis: *Ver/Alterar os detalhes da Sala, Apagar a Sala*
- **/room/< ID >/sensors (/full)* -** *Serve para que um utilizador possa aceder/gerir os sensores associados a uma sala com o identificador interno "ID";*
  Ações Possíveis: *Ver/Adicionar/Remover Sensores da Sala*
  (/full) - Opção opcional que apenas serve para que o endpoint devolva toda a informação de cada sensor

Sensors

- **/sensors -** *Devolve todos os sensores a que os utilizadores da API têm acesso*
- **/sensor -** *Serve para um administrador criar um novo sensor na API com os detalhes especificados*
- **/sensor/< ID > (/key)* -** *Serve para que os utilizadores possam gerir/aceder aos detalhes de um determinado sensor com o identificador interno "ID;*
  Ações Possíveis: *Ver/Alterar os detalhes do Sensor, Apagar o Sensor*
  (/key) - Opção opcional para o endpoint devolver a chave de encriptação do sensor
- **/sensor/< ID >/measure/< Opção > -** *Serve para que os utilizadores possam aceder às medições de um sensor com o identificador interno "ID" segundo uma determinada "Opção";*
  Ações Possíveis: *Obter a última medição, Obter todas as medições num dado intervalo, Obter a média das medições num intervalo*

Types

- **/types -** *Devolve todos os tipos de sensores a que os utilizadores da API têm acesso*
- **/type -** *Serve para os administradores criarem um novo tipo de sensores na API com os detalhes especificados*
- **/type/< ID > -** *Serve para que os utilizadores possam gerir/aceder aos detalhes de um determinado tipo de sensores com o identificador interno "ID;*

Ações Possíveis: *Ver/Alterar os detalhes do Tipo de Sensor*, *Apagar o Tipo de Sensor*

Users

- **/users (/full)* -** *Devolve todos os Utilizadores da API, apenas aos Administradores*
  (/full) - Opção opcional que apenas serve para que o endpoint devolva toda a informação de cada Utilizador
- **/user/< ID > -** *Serve para que os utilizadores possam gerir/aceder aos detalhes de um determinado Utilizador da API com o identificador interno "ID;*
  Ações Possíveis: *Ver detalhes do Utilizador, Atribuir/Retirar o papel de Administrador, Remover Utilizador*

Access Policies

- **/accessPolicies -** *Devolve todos as políticas de acesso aos recursos definidas na API, apenas aos Administradores*
- **/accessPolicy -** *Serve para os administradores criarem uma nova política de acesso com base nas condições enviadas*
  Ações Possíveis: *Combinação de Restrições: a nível de um grupo/um utilizador ; a nível de um determinado recurso/conjunto de recursos ; a nível de um determinado contexto (ex. horas) ; a nível da ação que vai ser realizada no recurso*
- **/accessPolicy/< ID > -** *Serve para que os utilizadores possam gerir uma determinada política de acesso com o identificador interno "ID";*
  Ações Possíveis: *Substituir a política existente por uma nova ; Remover a política de acesso*

4.3.3 Base de dados auxiliar

Na implementação, para além de se querer garantir que as medições de cada sensor são acessíveis remotamente, também se deve permitir que gestores da plataforma possam organizar os mesmos em salas e ainda atribuir-lhes tipos para poderem ser especificadas unidades de medida nos valores obtidos.
Além disso, guarda-se, também, os utilizadores da API para persistir e, posteriormente, averiguar se são administradores da plataforma ou não.

Deste modo, esta base de dados é relacional (em PostgreSQL), sendo esta constituída por apenas quatro tabelas:

- **Espaço** - *Esta entidade serve para representar o meio físico que envolve um conjunto de sensores, sendo que a maior parte das vezes vai ter elementos que são salas, no entanto podemos também representar um corredor por exemplo.*
  Cada elemento têm como campos, um nome e uma descrição

- **TipoSensor** - *nesta entidade, representa-se cada um dos tipos de dados que os sensores podem medir.*
  Os Tipos de Sensores apenas podem ser eliminados caso não exista mais nenhum sensor associado - caso contrário teria-se sensores com medições sem unidades (logo sem utilidade).
  Cada elemento tem também como campos um nome e uma descrição.

- **Sensor** - *representa cada instância de sensor, ou seja cada dispositivo existente, mesmo que se tenha dois exatamente iguais em salas diferentes.*
  Caso o sensor não tenha sala na base de dados, é considerado que se encontra em armazém, sendo que, quando se elimina uma sala onde ainda existem sensores, é considerado que são todos guardados em armazém.
  Cada elemento tem como campos uma Descrição, o nome do tipo dos tipos de dados medidos pelo sensor (representa o tipo de sensor), a unidade de medição em que os valores estão a ser medidos e ainda a sala em que este se encontra.

- **Utilizador** - *Esta entidade representa cada utilizador da API que tenha realizado a autenticação bem sucedida pelo menos uma vez.*
  Cada utilizador tem como campos o Email do utilizador e se se trata de um administrador ou não.

4.3.4 Controlo de acessos

A API combina regras internas pré-definidas de acesso administrativo a endpoints e a utilização um submódulo de controlo de acessos baseado em atributos (ABAC), desenvolvido em Python. Este utiliza uma framework denominada Vakt, que define um conjunto de funcionalidades capazes de fornecer uma solução de controlo de acesso por políticas baseada em dicionários Python e estruturas do tipo JSON, que unifica o desenvolvimento da API no que toca às mensagens transmitidas em todo o fluxo de dados com as componentes de *frontend*.

Regras internas de acesso

A especificação de endpoints da API reúne um conjunto de pressupostos que são internos e que definem, à partida, o acesso que deve ser dado em endpoints.

No que toca a endpoints que manipulam internamente recursos como utilizadores, sensores, salas e tipos de métricas, são definidos logo à partida como endpoints apenas para administradores: antes de processar o pedido HTTP para um destes endpoints, é verificado, na base de dados interna de atributos, se o utilizador que o fez é administrador. No caso de não o ser, é-lhe respondido que o acesso ao endpoint é interdito, ou seja, é-lhe enviada a resposta HTTP ao pedido efetuado com o código de estado 403 (*Forbidden*). Assim, é feito um pré-processamento leve deste pedido, ao invés de processar o endpoint, tratar erros expectáveis a falhar e pedidos desnecessários à camada de controlo. Portanto, o estabelecimento desta
regra interna na API é justificável.

Outra regra define, de uma maneira semelhante, que um utilizador só consegue aceder a qualquer endpoint se estiver autenticado, com exceção aos endpoints relacionados com o estabelecimento dessa sessão.

Dentro do motor de controlo de acessos, denominado *PolicyManager*, na primeira inicialização, é criada automaticamente uma política que define que administradores têm acesso a toda a API, se esta não existir. Isto é necessário porque o PDP (Policy Decision Point) - nome da classe implementada para o cliente efetuar consultas - rejeita todos os pedidos por omissão. Como, dentro dos utilizadores da plataforma, os administradores devem ter a mais alta prioridade, esta política deve estar sempre criada. Depois, é possível retirar acesso a outros administradores com a criação de outras políticas para fazer a restrição necessária.

Formato de políticas

No que toca à definição de uma política, esta deve ser uma mensagem JSON com entidades ou partes que devem ser processadas e, depois, persistidas de maneira a que, após a criação, pedidos HTTP sejam mapeados para um formato de uma consulta e enviados ao motor de controlo de acessos, em JSON, por forma a correspondê-los a políticas existentes e, consequentemente, retornar o resultado final para concessão ao acesso do recurso.

Na utilização do *toolkit*, é necessário considerar cinco entidades-chave
em cada mensagem.

- **Recursos** (*Resources*) - atributos que permitem definir recursos do sistema (O *que* se está a ser acedido?). A sua inclusão na criação de uma

política não é obrigatória: por defeito, é aplicada a qualquer recurso. É dada por uma lista de dicionários.

- **Sujeitos** (*Subjects*) - atributos relativos à caracterização dos utilizadores da qual tenta aceder ao recurso (*Quem* acede aos recursos?). A sua inclusão numa política é obrigatória, pois a identidade do indivíduo que acede a determinado recurso é crucial no sistema (como explicitado acima, todos os serviços se apoiam na autenticação com o IdP da UA, pelo servidor de OAuth). O seu valor deve ser uma lista de dicionários.
- **Ações** (*Actions*) - atributos que definem o que vai ser realizado com o recurso (*O que se pretende fazer* com o pedido de acesso aos recursos?). A criação de uma política não necessita da sua definição, sendo que, por omissão, a política engloba todas as ações.
- **Contexto** (*Context*) - atributos que se relacionam com as circunstâncias de acesso (*Em que contexto* é que esta política é satisfeita?). A sua definição não é obrigatória, sendo que, nesse caso, a política abrange pedidos de qualquer contexto.
- **Efeito** (*Effect*) - Se a política se destina a satisfazer o pedido pela positiva (*allow*) ou pela negativa (*deny*). Não sendo obrigatória para definir uma política, por defeito, o efeito é dado como *allow*. Definido como uma *string*.

É possível, ainda, incluir uma **Descrição** (*Description*) na política. Recomenda-se o uso desta entidade para se conseguir identificar as políticas de uma forma fácil e legível ao administrador, sem ter de analisar a política em si.

De notar que um dicionário dentro de cada parte inclui atributos, sendo que para fazer corresponder a uma consulta, **todos** os atributos dados devem ser satisfeitos dentro do dicionário. A lista define um conjunto em que, para o pedido satisfazer a política numa consulta, **um** dos atributos ou elementos desta deve ser correspondido. Portanto, um dicionário pode ser visto como um elemento individual de cada entidade ou parte (como um recurso nos recursos dados), em que a política é satisfeita se **qualquer um** elemento da lista de entidades fizer corresponder. Assim, cada elemento das partes são avaliadas em **OR lógico**, e os atributos dentro dos dicionários são avaliados posteriormente com um **AND lógico**.

Na criação de uma política, este formato interno é validado, sendo que um erro é assinalado com o motivo para a falha (é retornado um tuplo com um booleano e mensagem de erro).

Atributos

Nesta implementação, foram consideradas simplificações impostas para o desenvolvimento das vistas da plataforma de gestão que manipulam as políticas,

de maneira a que a experiência da utilização destas seja satisfatória. Assim, no que toca ao conjunto de atributos a avaliar em cada pedido, este foi reduzido em comparação com o leque de atributos que se é possível suportar, das fontes de informação disponíveis (atributos de utilizador dados pelo servidor de OAuth, atributos do pedido HTTP e atributos de contexto do pedido).

Com isto, foram considerados os seguintes atributos para avaliação, por cada elemento da entidade para definição numa política:

Entidade	Atributo	Valor	Descrição
Recursos (*resources*)	sensor	String (UUID)	ID interno de um sensor
	room	String (UUID)	ID interno de uma sala
	type	Inteiro	ID interno de um tipo de métrica
Sujeitos (*subjects*)	email	String	Email institucional da UA
	admin	Booleano	Administrador do sistema
	student	Booleano	Estudante na UA
	student_courses	Lista de inteiros	Códigos de unidades curriculares inscritas do estudante
	teacher	Booleano	Docente na UA
	teacher_courses	Lista de inteiros	Códigos de unidades curriculares lecionadas pelo docente
Ações (*actions*)	-	Lista de Strings: 'GET', 'POST', 'DELETE'	Métodos HTTP
Contexto (*context*)	day	Dicionário *from*: String *to*: String	Intervalo de datas no formato 'MM/DD/AAAA' ou 'AAAA/MM/DD'.
	hour	Dicionário *from*: String *to*: String	Intervalo de hora do dia no formato 'HH:MM:SS'
	ip	String: 'internal' ou 'external'	IP interno (interno apenas) ou externo (interno e externo)
Efeito (*effect*)	-	String: 'allow' ou 'deny'	Conceder o pedido ou rejeitá-lo
Descrição (*description*)	-	String	Descrição de uma política

Tabela 4.5: Atributos considerados nas políticas, por entidade

Criação da política

Na criação de uma política, a mensagem é validada em termos de sintaxe geral de JSON. No caso de ser válida, começa o processamento de cada entidade/parte da política para ser serializada e persistida como uma *Policy*, que é o objeto utilizado pelo framework. Em cada entidade desta, para cada atributo, são utilizadas funções internas, dadas pela ferramenta, para a criação de regras ('*Rules*') que serão avaliadas pelo motor aquando de um pedido que lhe origine uma consulta (ou seja, avaliadas pelo PDP). As seguintes regras são persistidas na base de dados, por entidade:

Entidade	Atributo	Regra de correspondência
Recursos (*resources*)	sensor	Igualdade da String
	room	Igualdade da String
	type	Igualdade
Sujeitos (*subjects*)	email	Igualdade da String
	admin	Veracidade / Falsidade
	student	Veracidade / Falsidade
	student_courses	Qualquer valor da lista da consulta na lista dada
	teacher	Veracidade / Falsidade
	teacher_courses	Qualquer valor da lista da consulta na lista dada
Ações (*actions*)	-	Atributo encontra-se na lista dada
Contexto (*context*)	day	Valor epoch dado em *from* <= Valor epoch na consulta <= Valor epoch dado em *to*
	hour	Valor dado em *from* (segundos) <= Valor na consulta (segundos) <= Valor dado em *to* (segundos)
	ip	Se 'internal', IP dado está dentro da rede interna (avaliação CIDR de qualquer das redes internas) Se 'external', qualquer IP

Tabela 4.6: Regras de correspondência numa política serializada

De notar que o atributo de sujeito '*admin*' é sempre incluído na política, de forma a fazer a priorização do atributo de administrador nas políticas. Assim, qualquer política criada que não contenha este atributo não afeta os administradores, a não ser que este seja dado como Verdadeiro.

Por fim, e assumindo que não existiu qualquer erro, à mensagem JSON não processada e original é adicionada um ID interno, e esta é guardada numa coleção de uma base de dados orientada a documentos, para conseguir ser legível e limpa na vistas de controlo de acessos da plataforma de gestão. A política processada e criada como um objeto é serializada e persistida, em JSON, com o mesmo ID interno criado anteriormente, numa coleção interna utilizada pela ferramenta de controlo de acesso. Estas duas coleções são sempre sincronizadas aquando da manipulação de uma política com as funções de atualização ou remoção, chamadas a partir dos endpoints específicos para o controlo de acessos.

Consulta (Inquiry)

Numa consulta, é utilizado um objeto do tipo PDP (implementado no módulo de controlo de acessos) para efetuar uma interrogação ao acesso a partir de um pedido HTTP. Numa chamada à função para o efeito, esta começa por obter os atributos de utilizador necessários, seguido da construção de uma Inquiry, que passa por fazer uma recolha e processamento de atributos correspondentes a cada uma das entidades/partes que são necessárias para definir o pedido feito do ponto de vista do motor de controlo de acessos. A tabela abaixo explica a obtenção dos valores de atributos na execução de uma consulta:

Entidade	Atributo	Valor
Recurso (*resource*)	sensor	Valor do sensor pedido, obtido pelo caminho do endpoint, como atributo opcional na chamada ou atributo de recursos na base de dados relacional
	room	Valor da sala pedida, obtido pelo caminho do endpoint, como atributo opcional na chamada ou atributo de recursos na base de dados relacional
	type	Valor do tipo de métrica pedido, obtido pelo caminho do endpoint, como atributo opcional na chamada ou atributo de recursos na base de dados relacional
Sujeito (*subject*)	email	Atributo de utilizador (Identity@UA ou cache)
	admin	Atributo de utilizador (base de dados relacional)
	student	Atributo processado de utilizador (Identity@UA ou cache): verificado se utilizador está inscrito em alguma UC
	student_courses	Atributo de utilizador (Identity@UA ou cache)
	teacher	Atributo processado de utilizador (Identity@UA ou cache): verificado se utilizador leciona alguma UC
	teacher_courses	Atributo de utilizador (Identity@UA ou cache)
Ação (*action*)	-	Método HTTP do pedido
Contexto (*context*)	day	Epoch da data no momento do pedido
	hour	Calculada a partir da data no momento do pedido
	ip	Primeiro IP do cabeçalho de proxy X-Forwarded (IP original do pedido)

Tabela 4.7 : Atributos incluídos numa consulta, por entidade

Considerações na API

Em termos de endpoints na API, averiguou-se algumas :

- Um utilizador que tenha acesso a um sensor específico, também tem acesso à sala e ao tipo de métrica desse sensor
- Um utilizador que tenha acesso a uma sala específica, tem acesso a todos os sensores da mesma sala
- Um utilizador que tenha acesso a um tipo de métrica específico, tem acesso a todos os sensores do mesmo tipo

Por isto, é necessário coordenar a classe PDP do módulo de controlo de acessos com implementações dos endpoints na API para que a análise de políticas na consulta faça corresponder estas regras:

- Um endpoint que liste salas ou tipos ou retorne metadados de uma sala ou tipo tem de verificar se o utilizador tem acesso a algum sensor correspondente.
- Endpoints relacionados com a obtenção de dados dos sensores têm sempre de incluir o tipo e a sala

4.3.5 Configuração do ambiente de execução

Configurações da API

A API remete variáveis globais de configuração da aplicação Flask, cruciais para o seu funcionamento, para um ficheiro de configuração no formato INI (*Initialization file*). Estas não podem ser colocadas diretamente no código por forma a que esta informação fique segura e de modo a que não seja visível para qualquer entidade. O ficheiro é oculto e apenas consegue ser acedido e alterado pelo utilizador que executa a API. Inclui:

- Chave e segredo de aplicação para comunicação da API com o servidor OAuth “Identity@UA”
- Strings de identificação das plataformas de frontend
- Chaves e configurações para criação e registo de sensores no backend pela API
- Configurações de utilizador de acesso ao broker agregador de dados sensoriais (Eclipse Hono).

Ficheiro de opções de ambiente

O ficheiro de opções de execução da API contém pares variável-valor que configuram os vários sub-módulos integrados. A grande diferença relativamente ao ficheiro de configuração apresentado anteriormente é a configuração de variáveis não estáticas no que toca ao contexto de execução, isto é, configuram acessos a serviços que podem ser desassociados, instalados ou clonados para outra localização remota - ou seja, as bases de dados. Este ficheiro deve denominar-se "*options.conf*" e deve ser escrito de maneira a apontar a API para os endereços corretos, de maneira a que a comunicação aos pontos de informação seja realizada com sucesso.

O formato do ficheiro é INI e deve ter o seguinte conteúdo:

```
[postgresql]
      URL = postgresql_ip
      PORT = postgresql_port
      DB = postgresql_database_name
      USER = postgresql_user
      PW = postgresql_password

[influxdb]
      URL = influx_ip
      PORT = influx_port
      DB = influx_database_name
      USER = influx_user
      PW = inlfux_password

[mongodb]
      host = mongo_ip
      port = mongo_port
      db = mongo_database_name
      collection = internal_policy_collection
      raw_collection = raw_json_policy_collection
```

Figura 4.11: Exemplo de configuração do ficheiro de opções.

Exposição por WSGI

A execução da aplicação Flask é feita a partir de um WSGI (Web Server Gateway Interface), que executa a API em várias threads e *workers* por thread, e cria uma interface que a expõe, intermediando a comunicação à API com os vários pedidos que lhe vão chegando, de uma forma assíncrona. Isto permite que o *throughput* agregado para os pedidos da API aumente significativamente, em pedidos processados por segundo, com o aproveitamento das CPU (Central Processing Unit) com múltiplos núcleos e múltiplas threads do servidor onde a aplicação está instalada.

O WSGI escolhido para intermediar as comunicações é o Gunicorn, pois é um WSGI Python capaz de integrar facilmente uma aplicação Flask, com recurso à mesma linguagem de programação.

No que toca à configuração realizada:

- O número de threads é livre, no entanto foi configurado que pode usar, no máximo, todas as threads disponíveis.
- Foi calculado o número de workers com base nas threads do CPU do servidor, sendo que existem, aproximadamente, dois workers por thread.
- Comunicações da camada de aplicação com o WSGI são seguras, ou seja, usa HTTPS, com certificados criados a partir de uma Autoridade Certificadora *self-signed*, que depois assina um certificado próprio para utilizar no WSGI. Assim, mesmo internamente, entre o proxy local e o WSGI/API, as comunicações são cifradas.

Na execução do WSGI, por cada worker, o inicializador executa uma instância da API Flask com os seus módulos internos, executando depois o seu *middleware*, que aponta para a localização base original da API.

4.4 Frontend

4.4.1 Plataforma de monitorização dos dados sensoriais

Dados e fontes de dados

Relativamente à plataforma para a monitorização e visualização dos dados provenientes dos sensores, foi escolhido o Grafana como ferramenta a ser utilizada, por ser uma solução open-source que permite uma monitorização eficaz de dados. Os primeiros passos a serem tomados para a monitorização de dados,foi a integração de dados dos sensores com o Grafana, usando como *data source* o InfluxDB, sendo desta forma obtidos os dados diretamente da base de dados onde são enviados os dados dos sensores.

Para a integração desta fonte de dados na plataforma, bastou apenas adicionar uma *data source* no Grafana, sendo necessário definir o endereço IP e porto da InfluxDB API onde são enviados os dados dos sensores, assim como, o nome da base de dados, credenciais de acesso e HTTP mode. Depois, foram criadas várias dashboards para os diferentes sensores, sendo os dados obtidos através de queries à base de dados, usando a sintaxe InfluxQL, bastante semelhante à sintaxe da linguagem SQL, inseridas na plataforma através da ferramenta Query Editor disponibilizada pelo Grafana.

Os dados podem ser formatados em modo de tabela ou *timeseries*, mas como o InfluxDB é uma base de dados *timeseries*, foi escolhido o último modo para a apresentação dos dados.

Figura 4.12: Query editor do Grafana

Após esta primeira fase, o próximo objectivo passou pela obtenção dos dados diretamente da API, em vez da base de dados, procurando-se assim a criação de uma camada de abstração, na qual a plataforma consome os dados

fornecidos pelos endpoints da API e o utilizador, por sua vez, não tem que inserir queries para obter os dados sensoriais. Como tal, foi utilizado o plugin Simple JSON Datasource que permite a obtenção de dados através da execução de pedidos JSON a endereços de *Backend* arbitrários.

Para a plataforma obter os dados diretamente da API, foi necessário implementar endpoints específicos, que fossem de acordo com aquilo que estava especificado na documentação do plugin. Para uma melhor organização da API, foi usado o conceito de *Blueprint* da framework Flask, sendo uma das utilidades deste conceito, a subdivisão da API em módulos, sendo então implementado um *Blueprint* para os endpoints relativos ao Grafana.

Foi então necessário implementar um endpoint para o teste da conexão da plataforma com a API, no momento de adicionar o *datasource* no Grafana. Por outro lado, houve a necessidade de implementar dois endpoints para a obtenção de dados diretamente da API: um endpoint que devolve a lista de todas as métricas (sensores) existentes e outro que devolve um conjunto de *data points* juntamente com as respetivas *timestamps*, relativos ao(s) sensor(es) alvo e ao intervalo de tempo pedido. De notar, que para não ser feito um envio em grande escala de dados sensoriais, os dados são agrupados em intervalos de 30 segundos, sendo feita a média desses valores. Foi ainda necessário a inclusão de um endpoint para comentários sobre intervalos de tempo no gráfico, que retorna um JSON vazio, uma vez que esta funcionalidade não é utilizada no projeto. Um resumo dos endpoints implementados pode ser encontrado na Tabela 4.8 abaixo:

Endpoint Base Path	Utilização
/	Usado para testar a conexão com a API
/search	Usado para devolver uma lista com os sensores presentes no sistema
/query	Usado para devolver os dados sensoriais juntamente dos *timestamps* respetivos, tendo em conta os campos presentes no pedido da plataforma que contém, entre outros, os campos relativos ao(s) sensor(es) alvo e intervalo de tempo
/annotations	Usado para devolver uma lista de anotações sobre um ou mais intervalos de tempo específicos no gráfico.

Tabela 4.8 : Resumo dos endpoints usados pelo grafana

Dashboards e painéis

Com a implementação dos endpoints necessários e respetivos testes para o consumo de dados através da API, a criação de dashboards e respetivos painéis revelou-se uma tarefa simples e direta. Para uma melhor organização da informação, foram criadas duas dashboards (uma para cada sala), em que cada uma contém vários painéis relativos aos sensores existentes nessa sala. Para a adição de um novo painel, basta selecionar um ou mais sensores da lista de sensores existentes, apresentados com o formato SalaX_SensorY (Metrica_Sensor), sendo esta lista obtida através do endpoint /search. Após esta seleção, os dados são automaticamente apresentados, passando os próximos passos por definir o intervalo de tempo e os aspetos visuais do painel. Um painel no Grafana apresenta vários modos de visualização como gráfico, tabela e heatmap, mas em todos os sensores optou-se por um gráfico como modo de visualização. Por fim, é possível optar pela configuração de alertas relativos ao painel, tema que irá ser abordado no subcapítulo seguinte.

Figura 4.13: Painéis sensoriais relativos à dashboard de uma sala

Alertas

A configuração de alertas no Grafana revelou-se como uma tarefa de extrema importância, pois permitiu alertar de forma quase instantânea os desenvolvedores de sistemas de IoT que utilizam o sistema, no caso de falhas ou anomalias nos seus sensores, e por outro lado, permitiu que estas servissem como *trigger* para o envio dinâmico de notificações push para a aplicação móvel.

O Grafana já disponibiliza por defeito a configuração de alertas para um painel, mas o plugin que permite o consumo de dados através de uma API não é compatível com o sistema de alertas do Grafana. Por isso, houve uma necessidade de fazer alterações no plugin disponibilizado, passando a solução por transformá-lo num Backend Plugin para permitir o suporte do sistema de alertas, sendo usado o trabalho desenvolvido numa branch do repositório GitHub do plugin como referência.

Após a incorporação do sistema de alertas no plugin, foi necessário configurar os canais de notificações, podendo estas notificações serem enviadas para o Email, Discord, Slack, Webhooks, entre outros sistemas que se encontram listados no site do Grafana. Foi precisamente com a configuração de um canal do tipo webhook que é feito o envio dinâmico de notificações push para a aplicação móvel. Para isso, foi necessário implementar um endpoint na API que funciona como um Webhook Listener, fazendo a leitura do conteúdo enviado nos alertas do Grafana e transformando esse conteúdo numa notificação FirebaseCM que é enviada para um tópico específico relativo ao sensor em questão.

O próximo passo, passou pela configuração nos painéis dos alertas, podendo a lista de alertas configurados e o seu estado ser consultado na tab de

Alerting, em *Alert Rules*. Os alertas no Grafana funcionam como uma máquina de estados sendo o estado normal o de OK, e no caso de violação de uma regra definida pelo utilizador, existe uma transição do estado de OK para Pending. Neste estado não é enviada logo a notificação, sendo a regra avaliada constantemente e no caso desta continuar a ser violada durante um intervalo de tempo definido pelo utilizador, é então feita a transição do estado Pending para Alerting, sendo enviada a notificação. Outro estado existente é o No Data, sendo feita a transição automática para este estado, no caso de não haver dados, ou estes serem todos nulos num painel. A configuração passa por definir uma ou mais condições que são avaliadas a cada intervalo de tempo definido, assim como um intervalo de tempo na qual esta deve estar no estado de Pending. São também definidos os canais de notificações a ser usados e o conteúdo da mensagem.

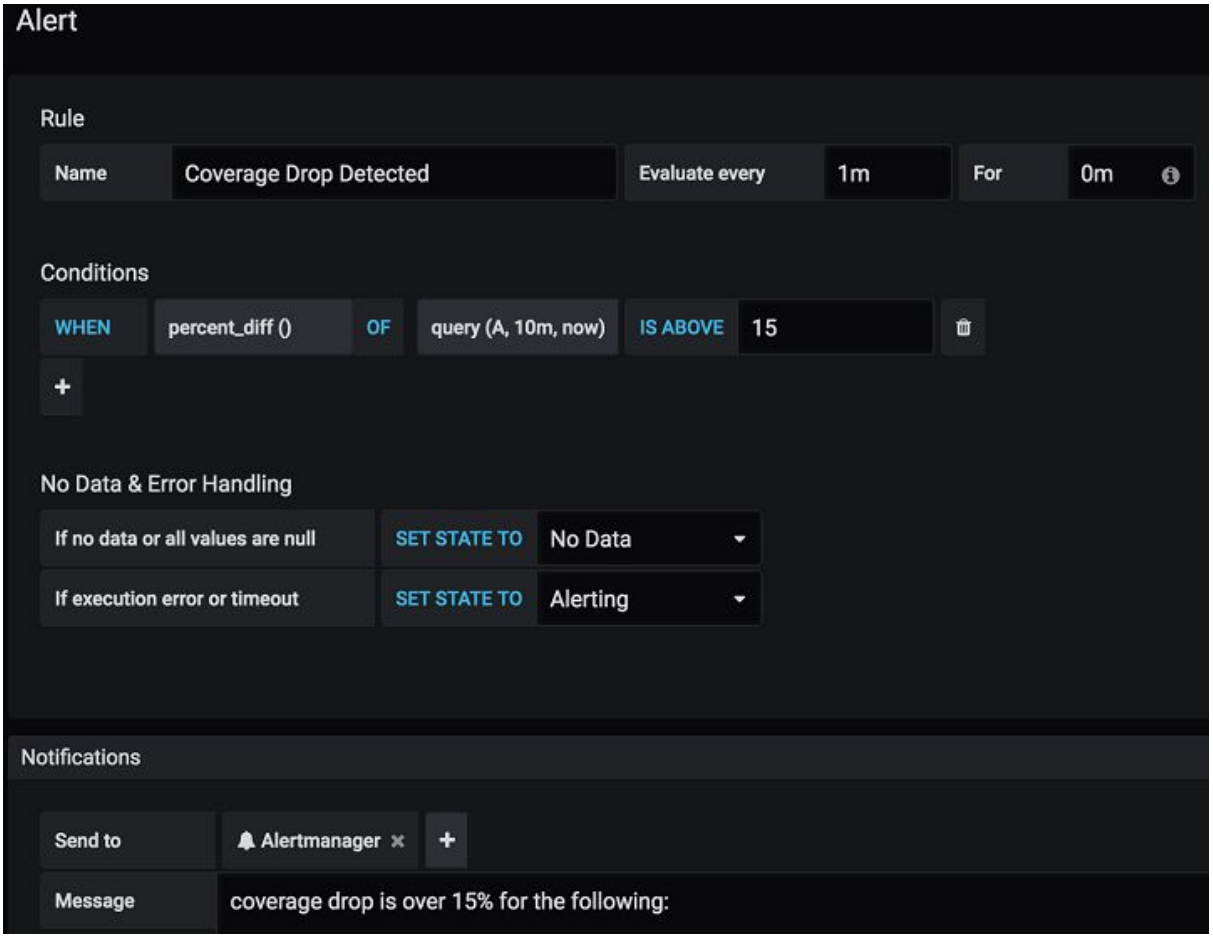

Figura 4.14: Exemplo de configuração de um alerta no Grafana

Login e acesso

A integração da autenticação na API com o Grafana foi realizada com recurso à funcionalidade de autenticação por um proxy de autenticação (*Auth Proxy*) que o serviço de *dashboards* disponibiliza.

Esta integração utiliza as capacidades de um proxy Nginx que expõe todos os serviços web internos existentes no servidor e um endpoint especial, "/wsauthverify", que associa uma resposta de validação de sessão por parte da API para a localização do proxy correspondente:

- O utilizador entra na localização que corresponde a um acesso ao servidor do Grafana. Este verifica o estado da sua sessão, com recurso ao endpoint de verificação. No caso deste já se encontrar autenticado na API, então este é, novamente, redirecionado automaticamente para a página inicial associada às dashboards.
- No caso de não estar autenticado, este é redirecionado para o endpoint de autenticação. O processo é efetuado e, no final, como já referido na secção de sessão autenticada, na explicitação dos endpoints da API, uma cookie especial de sessão é criada para o efeito. A API, por fim, redireciona-o para a página principal das dashboards.

De notar que o acesso à plataforma de dashboards é apenas admitida a administradores.

4.4.2 Aplicação Móvel

A aplicação móvel associada ao projecto tem por objectivo principal fornecer aos utilizadores comuns acesso aos dados dos sensores instalados, de forma conveniente e rápida.

Devido à sua capacidade de criação de aplicações de forma rápida e de fácil compreensão, foi utilizada a framework Flutter, recorrendo, também a vários plugins da mesma.

As principais funcionalidades implementadas foram:

- Listagem de todas as salas monitorizadas pelo sistema
- Disponibilização dos dados sensoriais registados numa dada sala, bem como avaliação dos mesmos
- Recepção de alertas e configuração de quais alertas a receber
- Fazer pedidos autenticados à API

Deste modo, o desenvolvimento da aplicação móvel pode ser dividido em três componentes fundamentais:

- Autenticação
- Aquisição e Apresentação de Informação Sensorial
- Sistema de Alertas

Autenticação

Ao entrar na aplicação, o utilizador é redireccionado para uma página de login. Foi utilizado um plugin externo para poder ser mostrado na aplicação a página do idp da universidade de aveiro, através da qual o utilizador efectua o login. O url passado a este plugin é o endpoint de autenticação da API, que quando chamado, redirecciona o utilizador para o idp. Uma vez introduzidas as credenciais, estas são validadas conforme explicado na secção referente à API.

Uma vez efectuado o login, a aplicação recebe um id de sessão que é usado para todos os pedidos subsequentes feitos à API.

Em qualquer altura, após estar autenticado, o utilizador pode fazer o logout na aplicação

Plugins utilizados	Utilização
http	Efectuar pedidos HTTP à API
webview_flutter	Apresentar a página do idp da universidade para login

Tabela 4.9 : Resumo dos Plugin utilizados na aplicação móvel para autenticação

Aquisição e Apresentação de Informação Sensorial

O aspecto mais importante da aplicação móvel é a possibilidade de visualizar os dados sensoriais de forma rápida e conveniente. Isto é possível a qualquer utilizador autenticado no sistema, que terá acesso aos dados dos sensores que lhe sejam permitidos.

Uma vez efectuado o login, o utilizador é levado para a página de listagem de salas. Ao iniciar, a aplicação efectua uma sequência de pedidos à API (ilustrada no Diagrama 4.15), com o objectivo de recolher informação sobre todas as salas registadas, os seus detalhes e informação sobre os sensores instalados nas mesmas.

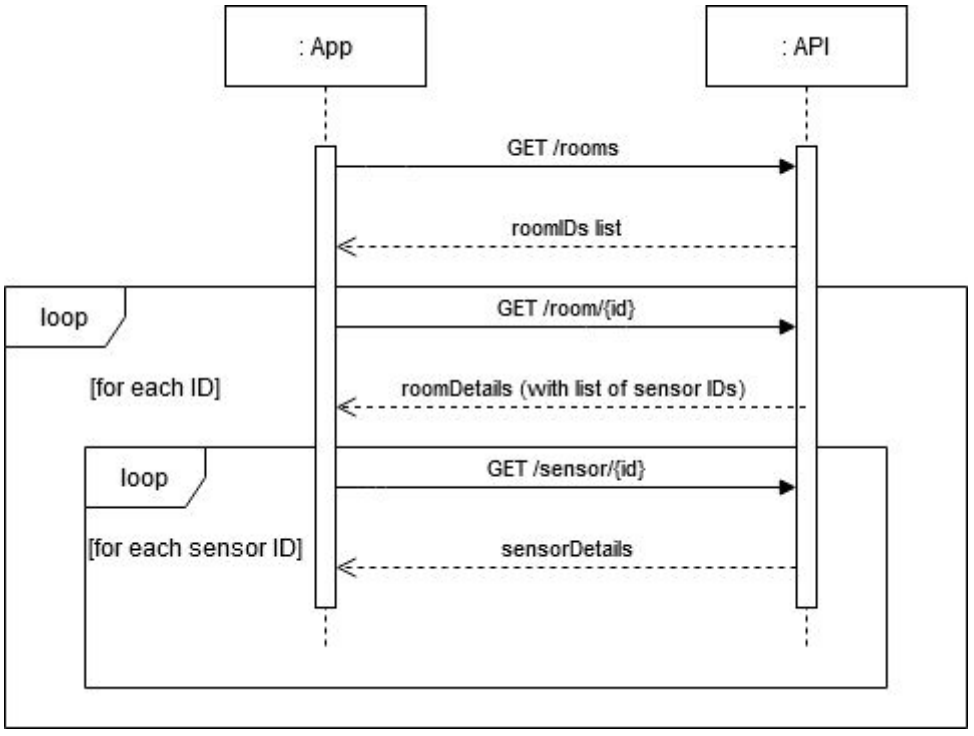

Figura 4.15: Sequência de aquisição dos dados das salas

Nesta altura, é possível, também procurar uma sala específica por nome. Por cada uma das salas listadas, o utilizador pode expandir para ter mais informações, provenientes da API, tais como uma descrição da sala e todos os sensores nela instalados.

Ao seleccionar uma sala, o utilizador é levado para a página de detalhes dos sensores, onde são listadas as informações sensoriais para cada sensor instalado. Para cada sensor, é efectuado um pedido à API, sendo recolhidos os dados do último minuto. Posteriormente, estes dados são apresentados num gráfico. É, ainda calculado o ganho em valor absoluto e percentagem. Finalmente, para cada métrica adequada, é avaliado o valor medido actualmente, num de cinco níveis, para indicar se o valor se encontra demasiado alto ou demasiado baixo.

A Figura 4.16 ilustra esta interface.

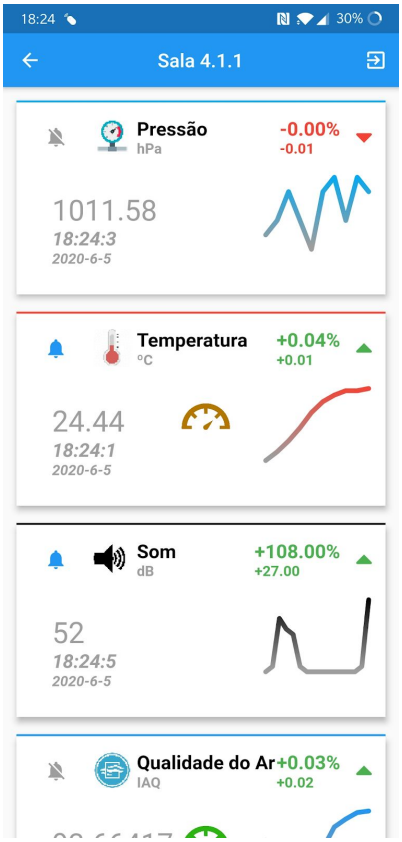

Figura 4.16: Página de detalhes sensoriais

Usando o sistema de alertas, é, também possível activar ou desactivar (encontrando-se desactivado por defeito) notificações para cada sensor.

Endpoints relevantes	Utilização
/rooms	Usado para preencher a lista de salas monitorizadas a que o utilizador tem acesso
/room/<id>	Obter os detalhes de uma sala identificada por <id>
/room/<id>/sensors	Obter a lista de ids de sensores para uma sala identificada por <id>
/sensor/<id>	Obter os detalhes de um sensor identificado por <id>
/sensor/<id>/measure/interval	Obtenção dos valores registados pelo sensor identificado por <id> ao longo do último minuto

Tabela 4.10 : Endpoints relevantes utilizados pela aplicação móvel

Plugins utilizados	Utilização
http	Efectuar pedidos HTTP à API
typicons_flutter	Utilização de diversos ícones relevantes
flutter_sparkline	Apresentação de informação sensorial em gráficos

Tabela 4.11 : Resumo dos plugins utilizados na aplicação móvel para apresentação dos dados

Sistema de Alertas

A aplicação móvel também pode receber alertas provenientes do sistema. Estes alertas são de dois tipos: alertas de controlo ou de sensor. Qualquer utilizador da aplicação recebe os alertas de controlo que são enviados, por exemplo, por um administrador. No entanto, o utilizador pode escolher sobre quais sensores pretende receber notificações.

Para poder receber os alertas de sensores, a aplicação liga-se ao projecto Firebase criado para o propósito. Quando o sistema envia os alertas, estes são, também, para o projecto Firebase referido, que, por sua vez envia no tópico do respectivo sensor que é identificado pelo seu ID. Se o utilizador pretender receber notificações de um dado sensor, deve-se subscrever ao tópico do sensor que pretende. Como foi dito, todos os utilizadores estão, sempre,

subscritos ao tópico de controlo. Na página de sensores de uma sala, ao lado de cada sensor, o utilizador pode activar ou desactivar os alertas do sensor em questão.

Devido à simplicidade da informação, optámos por guardar a lista de sensores subscritos na memória interna do telemóvel, num ficheiro, contendo os ids dos mesmos.

Para os dois tipos de alertas, uma vez recebidos pelo Firebase, são reenviados para a aplicação, que apresenta uma notificação Android no telemóvel.

Na figura 4.17 apresenta-se um exemplo de uma notificação de sensor e uma de controlo.

Figura 4.17: Exemplos de notificações

Plugins utilizados	**Utilização**
firebase_messaging	Interacção com o projecto Firebase
flutter_local_notifications	Apresentação dos alertas numa notificação android
path_provider	Identificar caminhos no sistema de ficheiros

Tabela 4.12 : Resumo dos plugins utilizados na aplicação móvel para as notificações

4.4.3 Plataforma de Gestão

Destinada apenas a administradores, a Plataforma de Gestão exemplifica algumas das inúmeras variantes possibilitadas pela API. Desde uma simples visualização das salas disponíveis, passando pela criação de tipos de métricas e pelo sistema de notificações para os utilizadores da Aplicação Móvel, finalizando na gestão das políticas de acesso aos diversos sensores, entre outros, a Plataforma de Gestão permite aceder e controlar a maioria da informação gerada.

A *framework* Django foi a escolhida, pelo facto de possibilitar a interligação do código desenvolvido em Python com a construção da página através de HTML, CSS e JavaScript. A passagem de parâmetros obtidos nos *requests* feitos pela biblioteca do Python para o próprio documento HTML foi um dos passos essenciais para desenvolver a Plataforma de Gestão com sucesso.

Página Inicial

Após ser autenticado com êxito na Plataforma de Gestão, o utilizador será redirecionado para a Página Inicial. Nesta, poderá visualizar os cinco atalhos para as secções referidas em seguida, bem como quatro referências para: Grafana, Documentação da API, Microsite desenvolvido para o projeto e o repositório do GitHub com o mesmo.

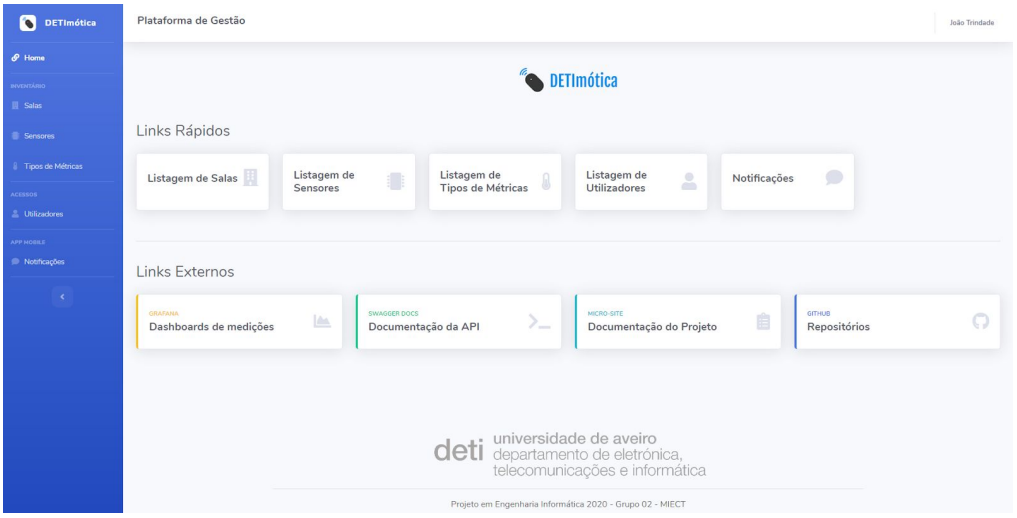

Figura 4.18: Exemplo da Página Inicial da Plataforma.

Salas

A página denominada “Salas” lista todas as salas disponíveis numa tabela onde é possível ver o UUID atribuído à divisão (gerado do lado do servidor), bem como a sua designação e descrição. Os UUIDs das divisões são obtidos recorrendo ao endpoint [GET] /rooms, com os metadados de cada sala a resultarem do [GET] /room/<UUID>.

É possível criar novas salas e editar os metadados da mesma. Clicando no botão “Adicionar”, preenchendo os campos “Nome” e “Descrição” e pressionando o ícone com a *checkmark*, o endpoint [POST] /room será chamado, contendo um JSON com os metadados fornecidos pelo utilizador e gerando um UUID que será atribuído àquela sala em específico. A tarefa de alterar as informações de uma sala em específico é bastante semelhante, recorrendo no entanto ao [POST] /room/<UUID> do respetivo repartimento.

Para além disso, existe ainda a hipótese de extinguir salas listadas, bastando para isso clicar no ícone vermelho que ativará o [DELETE] /room/<UUID>.

Figura 4.19: Exemplo da criação de um Sala na Plataforma de Gestão.

Sensores

Na secção sensores começa por ser apresentada uma lista das salas disponíveis, contendo apenas a nomenclatura de cada uma. Esta escolha baseia-se no facto dos sensores estarem organizados por divisões. À semelhança da lista de salas referida anteriormente, são obtidos os UUIDs de cada sala através do [GET] /rooms, sendo depois efetuado um novo [GET] /room/<UUID> para cada UUID obtido e apresentado o “Nome” associado ao mesmo.

Figura 4.20: Exemplo da primeira página apresentada na secção Sensores.

Ao clicar no compartimento que pretende, será apresentada uma nova página onde é possível visualizar todas as informações disponíveis da própria divisão, bem como dos sensores contidos na mesma. É ainda possível efetuar diversas ações sobre os mesmos.

Esta segunda página recebe o UUID da sala em forma de parâmetro e um [GET] /room/<UUID> devolve os metadados da mesma, permitindo assim apresentar os detalhes da divisão. Para além do [POST] /room/<UUID> com as informações transmitidas num JSON para editar a sala e do [DELETE] /room/<UUID> para apagar a mesma, é ainda possível aceder à página das políticas de acesso do compartimento em si. Uma generalização da gestão das políticas de acesso pode ser encontrada aqui.

Já os sensores, são apresentados numa tabela que possui o seu UUID, o "Tipo de Métrica", as "Unidades" relativas ao Tipo de Métrica e a "Descrição" associada ao aparelho. Primeiramente, é efetuado um [GET] /room/<UUID>/sensors que retorna uma lista com todos os UUIDs dos dispositivos associados à sala em questão. Posto isso, é efetuado um [GET] /sensor/<UUID> para cada sensor da lista obtida que devolve então as informações necessárias para preencher a tabela.

Analogamente ao que acontece na tabela das salas, é permitido adicionar, alterar e remover um sensor. O primeiro ocorre através do [POST] /sensor e tem o UUID gerado de forma automática, com os detalhes a serem novamente transmitidos no formato JSON, tal como no [POST] /sensor/<UUID> em que se alteram os metadados de um sensor. Para remover algum dos aparelhos listados, é chamado o endpoint [DELETE] /sensor/<UUID>, removendo então o sensor com sucesso.

Um dos campos de preenchimento obrigatório na criação e/ou alteração de um dispositivo é o "Tipo de Métrica". Esse campo é apresentado por um *dropdown* que contém todas as métricas disponíveis, métricas essas carregadas através do [GET] /types, seguido de um [GET] /type/<ID> para cada tipo de métrica.

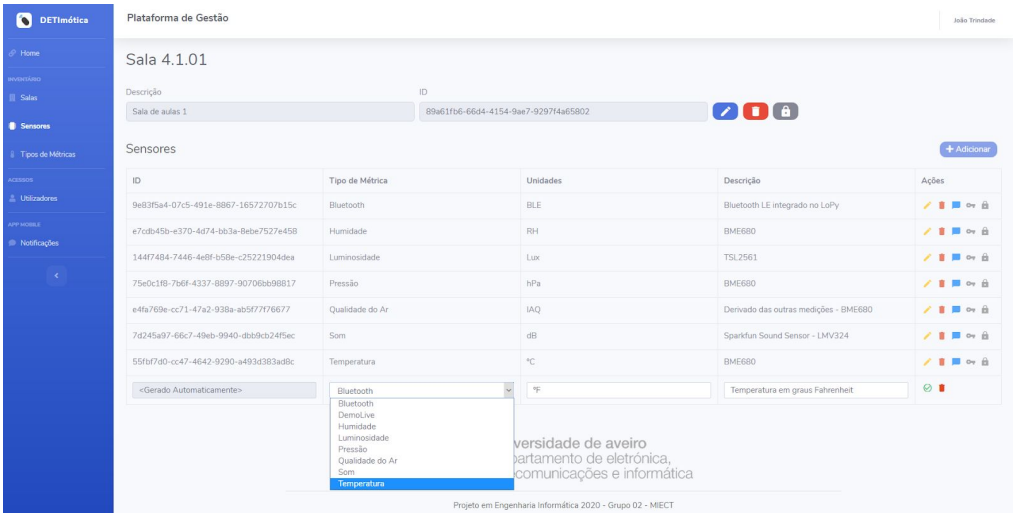

Figura 4.21: Exemplo da criação de um Sensor na Plataforma de Gestão.

Na figura acima, é possível visualizar três ações extras que não se encontram na tabela das salas, sendo que a terceira está associada à página das políticas de acesso do sensor. A primeira, representada por um ícone de uma notificação, permite enviar uma mensagem composta por "Assunto" e "Corpo". Essa mensagem será recebida por todos os utilizadores da Aplicação Móvel que subscreveram o sensor em questão. Ao contrário dos outros endpoints que efetuam ações do tipo [POST], para além das componentes da mensagem, também o UUID do sensor é enviado no JSON no corpo do [POST] /mobile/notifications.

A ação seguinte é relativa à chave de encriptação, revelando a mesma ao pressionar o ícone. Para obter a chave de encriptação AES-128, é chamado o endpoint [GET] /sensor/<UUID>/key.

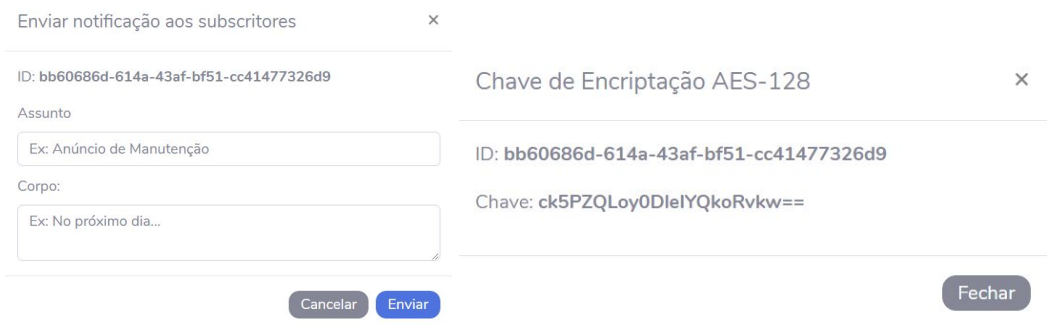

Figura 4.22: Exemplo do formulário para enviar uma notificação e da chave de encriptação de um Sensor.

Tipos de Métricas

Relativamente aos tipos de métricas, o processo é semelhante ao efetuado na página das salas. É primeiramente efetuado um [GET] /types que devolve uma *array* com os IDs de todos os tipos de métricas disponíveis. Percorrendo todos os elementos da lista, são efetuados diversos [GET] /type/<ID> para obter os metadados de cada tipo de métrica, nomeadamente o seu nome, descrição e unidades.

É então construída uma tabela com os metadados obtidos, sendo permitido adicionar, alterar ou ainda apagar um tipo de métrica na mesma. Os campos preenchidos ao criar ou alterar uma métrica são passados através de um JSON nos endpoints [POST] /types e /type/<ID>, respetivamente, com o primeiro a obter um ID criado automaticamente. De notar que o campo das unidades não é passível de alterações diretas, pois este é completo pelo conjunto de unidades de todos os sensores referentes àquele tipo. Ao clicar no ícone de remoção de um tipo, acionará o [DELETE] /type/<ID>, excluindo o mesmo.

Tal como nos sensores e nas divisões, é também possível criar diferentes políticas de acesso para cada tipo de métrica. O terceiro e último ícone redireciona o utilizador para a página de gestão das mesmas.

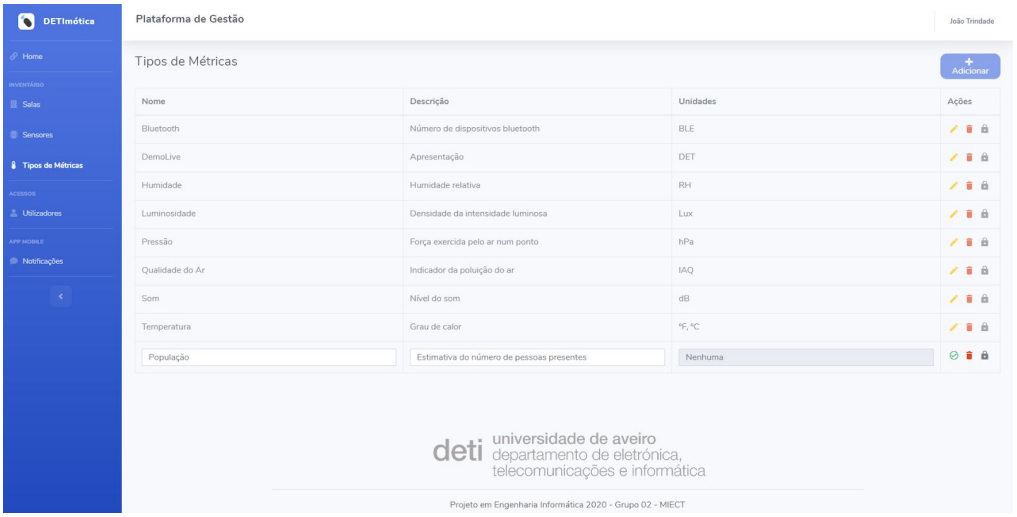

Figura 4.23: Exemplo da adição de um novo Tipo de Sensor na Plataforma de Gestão.

Políticas de Controlo de Acessos

Como referido na secção dos Sensores e dos Tipos de Métricas, é possível definir e gerir políticas de acesso para salas, sensores e tipos de métricas. Primeiramente, é executado um [GET] /<room/sensor/type/>/<UUID/ID> para obter os metadados do alvo a que queremos aplicar ou modificar uma ou várias políticas de acesso. É assim possível exibir o ID no caso dos sensores e a nomenclatura no caso das salas e tipos de métrica no início da página.

Para obter todas as políticas de acesso associadas a um determinado objeto (sala, sensor ou tipo de métrica), é convocado o endpoint [GET] /accessPolicies com o objeto e o UUID ou ID pretendidos a serem transmitidos por JSON, filtrando assim as políticas relevantes.

Não destoando das restantes páginas, as ações de adicionar e de alterar uma política de acesso utilizam o [POST] /accessPolicy e [POST] /accessPolicy/<UUID>, com os detalhes das mesmas no corpo do pedido. No primeiro caso, um UUID é atribuído à política no momento da sua criação. Já para remover uma restrição de acesso, o [DELETE] /accessPolicy/<UUID> é o endpoint designado para a tarefa.

Figura 4.24: Exemplo de uma página de gestão das Políticas de Controlo de Acessos.

O formulário para restringir ou alargar o acesso de uma política está dividido em cinco áreas: "Sujeitos", "Contexto", "Descrição", "Efeito" e "Métodos". O último é referente ao tipo de pedidos que serão alvos da política, podendo estes ser GET, POST e/ou DELETE. O "Efeito" tem apenas duas opções, sendo estas a de permissão ou proibição, caso queira alargar ou restringir o acesso, respetivamente.

A secção do "Contexto" é composta por três campos, sendo um deles o "IP", com as hipóteses "Interno" e "Externo". Ao escolher a segunda, não existirá qualquer restrição a nível de endereços visto serem permitidos IPs externos. Já a opção "Interno", bloqueará qualquer IP exterior à rede da Universidade de Aveiro, sendo por isso necessário estar presente na rede da mesma, seja presencialmente ou utilizando, por exemplo, a VPN da Universidade de Aveiro.

Os outros dois campos são o "Dia" e a "Hora", sendo possível combinar os dois para impedir ou permitir diariamente, durante o intervalo de dias estabelecido, o acesso à sala/sensor/tipo de métrica em causa, ao longo das horas pré-definidas. De notar que as horas determinadas estarão no fuso horário UTC, e que, o final do intervalo de dias é exclusivo.

No que diz respeito aos "Sujeitos", é possível especificar os mesmos de até quatro maneiras diferentes. A primeira opção, passa por determinar se os "Administradores" devem ou não ser alvos da política pretendida. As duas seguintes, "Docentes" e "Estudantes", quando ativas, bloqueiam ou permitem todos os docentes e estudantes por *default*. É, no entanto, possível especificar a(s) cadeira(s) que queremos no escopo da política, introduzindo o(s) código(s) da(s) mesma(s) na caixa de texto respetiva. Já o campo "Email" permite associar emails à política, vinculando assim indivíduos e não grupos à mesma.

Figura 4.25: Exemplo de uma Política de Controlo de Acesso na Plataforma de Gestão.

Utilizadores

Qualquer utilizador que aceda ao endpoint /login ficará registado. É possível fazê-lo diretamente, entrando na Aplicação Móvel, ou tentando ingressar na Plataforma de Gestão e/ou Grafana, com estes dois últimos a recusarem a autenticação de qualquer pessoa que não tenha o estatuto de administrador.

Para obter uma lista de todos os utilizadores que já efetuaram o login, bem como as suas informações, recorreu-se ao [GET] /users/full. Estas informações contêm o email e um booleano denominado "admin" que declaram se o utilizador em questão é ou não um administrador do sistema. Para além do "ID", "Email" e do "Admin", a tabela presente nesta página é composta ainda por uma coluna com um botão que permite alternar o valor do booleano.

É assim possível visualizar todos os utilizadores que têm acesso à plataforma visto esta estar apenas disponível a administradores.

Figura 4.26: Exemplo da página Utilizadores da Plataforma de Gestão.

Notificações

Para além das notificações referentes a um sensor em específico, o sistema de notificações possui uma página reserva apenas para o mesmo. Nela é possível preencher dois campos, o "Assunto" e o "Corpo", tal como nas notificações dos sensores, mas existindo uma grande diferença: esta página não tem qualquer UUID ou ID associado. Sendo assim, a notificação é enviada através do [POST] /mobile/notifications com o seu JSON a conter apenas as duas secções da mensagem.

Todos os utilizadores da Aplicação Móvel receberão, por isso, quaisquer notificações enviadas a partir desta página, sendo idealizada para alertas e informações gerais.

Figura 4.27: Exemplo da página de Notificações da Plataforma de Gestão.

5 Resultados e Discussão

No geral, todos os objetivos principais do projeto foram cumpridos, já que as respetivas funcionalidades foram implementadas e validadas através de demonstrações e testes exaustivos. Da mesma forma, as integrações entre os diversos módulos foram igualmente realizadas e validadas.

Na **secção 5.2** são apresentados e discutidos os objetivos e casos de uso que não foram possíveis de cumprir. Por outro lado, na **secção 5.3** enumera-se algumas funcionalidades desenvolvidas que não faziam parte da planificação original.

5.1 Exatidão dos sensores instalados

Os valores obtidos pelos sensores das métricas *Temperatura*, *Humidade*, *Pressão*, *e Dispositivos BLE visíveis* foram confirmados através de comparação direta com sensores comerciais equivalentes.

	BME 680 (DETImótica)	Geonaute RTGR328N (Controlo)
Temperatura	21.1ºC	20.6 ºC
Humidade	75 %RH	70 %RH
Pressão	1018.2 hPa	1018.9 hPa

Tabela 5.1: Medições de controlo efetuadas ao sensor BME 680

Foi também averiguado que o sensor de ruído não se encontra corretamente calibrado, visto apresentar uma sensibilidade bastante reduzida. No entanto, esta situação não pode ser resolvida devido aos motivos expostos na secção seguinte.

A exatidão dos valores das restantes métricas (Concentração de gases VOC e Luminosidade) não pode ser confirmada devido à falta de equipamento especializado para o efeito por consequência da suspensão das atividades letivas presenciais.

5.2 Objetivos que não foram possíveis de concluir

A integração com o API Gateway dos sTIC (WSO2) foi abandonada devido à falta de atributos providenciados pelo mesmo sobre o utilizador, especialmente quando comparado à integração direta com o identity.ua.pt.
Este aspeto é importante devido à sua relevância para o sistema de políticas de controlo de acessos.

Dada a suspensão das atividades letivas presenciais na Universidade de Aveiro, que se iniciou no dia 12 de Março, passou a existir uma grande dificuldade no acesso a componentes eletrónicos específicos e material de auxílio que normalmente está disponível no departamento (OpAmps, solda, entre outros).

Esta situação causou os seguintes entraves:

- O sensor de som necessita de ser calibrado, requerendo a soldagem (na qual nenhum dos elementos tem acesso às ferramentas necessárias) de uma resistência ou de um potenciómetro no circuito integrado do sensor, de modo a alterar o ganho do sensor, isto é, alterar a sensibilidade do sensor ao ruído.
  De momento, este encontra-se bastante pouco sensível pelo que os dados obtidos não podem ser considerados para a realização de uma estimativa fidedigna.

- A montagem do sensor de concentração de gás dióxido de carbono requer utilização de um OpAmp e componentes eletrônicos adicionais. Os membros do grupo encomendaram um exemplar mas não foi possível concluir a montagem sem o material adequado.
  No entanto, o código-fonte foi desenvolvido e encontra-se no repositório.

- Os medidores de corrente elétrica permaneceram no DETI devido à sua natureza e ao facto de que o plano original era a sua instalação ser efetuada por eletricistas profissionais.

O algoritmo de estimativa de ocupantes numa sala não pode ser implementado devido à insuficiência de dados relevantes causada pela falta dos sensores supracitados e o estado de calibração do sensor de ruído.

5.3 Funcionalidades extra desenvolvidas

- Integração de alertas provenientes do Grafana com plataformas externas.
- Sistema de notificações manuais para todos os utilizadores da aplicação móvel.
- Desenvolvimento de uma plataforma que permite uma gestão completa das políticas associadas aos sensores, salas e tipos de métricas, para além de permitir editar os metadados dos mesmos.
- Informação sobre o valor medido, simbolizando um aumento ou diminuição relativa à medição anterior, na aplicação móvel.

6 Conclusão

Para concluir o nosso projeto teve como principal objetivo o desenvolvimento de uma interface programática que permitisse o acesso remoto aos dados de sensores instalados no Departamento e ainda possibilitasse a gestão e a inclusão de novos sensores.

Neste sentido, foram programados vários sensores e foram instalados de forma semelhante à que se tivessem sido no nosso Departamento. Foi desenvolvido igualmente a interface programática e os módulos de base de dados, tendo sido tudo isto posteriormente integrado com sucesso.

Para a gestão dos sensores e dos acessos aos mesmos foi desenvolvida também por completo uma plataforma de gestão.

Por fim para prova de conceito ainda desenvolvemos uma aplicação móvel e um *dashboard web.*

Quantos aos resultados, o nosso projeto veio principalmente permitir que o processo desde a instalação de um sensor, até à sua disponibilização fosse encurtado, visto que neste momento basta após termos equipado, por exemplo uma sala, registar na nossa plataforma os novos sensores e a sala, e por fim usando a interface simplesmente selecionar que utilizadores devem aceder e de que modo o podem fazer, com base na definição de uma regra.

Como objetivos futuros, seria interessante acrescentar na interface programática um módulo de monitorização para a obtenção de estatísticas; melhorar as notificações para que pudessem ser personalizadas pelos utilizadores (por exemplo estabelecer limites com base na medição de um sensor); Incorporar um algoritmo de inteligência artificial em câmaras de vídeo, cujo intuito seria detectar e estimar o número de pessoas presentes numa determinada divisão.

7 Referências

Universidade de Alicante. n.d. UA Smart University. Universidade de Alicante. Retirado a 6 de junho, 2020 de https://web.ua.es/en/smart.

Bertyl Lankhaar. 2016. Living Smart Campus. Universidade de Twente. Retirado a 6 de junho, 2020 de https://www.utwente.nl/en/organisation/news-agenda/special/2016/living-smart-campus.

Pycom. 2018. LoPy v1.0 Datasheet. Datasheet de autoria de Pycom Go Invent. Retirado a 6 de junho, 2020 de https://pycom.io/wp-content/uploads/2018/08/lopy-specsheet.pdf.

Texas Advanced Optoelectronic Solutions Inc. 2005. TSL2560, TSL2561 Light-To-Digital Converter. Datasheet de autoria de Texas Advanced Optoelectronic Solutions Inc. Retirado a 6 de junho, 2020 de https://cdn.sparkfun.com/datasheets/Sensors/LightImaging/TSL2561.pdf.

Fairchild Semiconductor Corporation. 2012. LMV321, LMV358, LMV324 General Purpose, Low Voltage, Rail-to-Rail Output Amplifiers. Datasheet de autoria de Fairchild Semiconductor Corporation. Retirado a 6 de junho, 2020 de https://cdn.sparkfun.com/datasheets/Sensors/Sound/LMV324.pdf.

DFRobot. n.d. Gravity I2C BME680 Environmental Sensor VOC, Temperature, Humidity, Barometer SKU SEN0248. DFRobot Wiki. Retirado a 6 de junho, 2020 de https://wiki.dfrobot.com/Gravity__I2C_BME680_Environmental_Sensor__VOC,_Temperature,_Humidity,_Barometer___SKU__SEN0248.

Zack Lalanne. 2018. Using Two MQTT Brokers with Broker Bridging. Self Hosted Home. Retirado a 6 de junho, 2020 de https://selfhostedhome.com/using-two-mqtt-brokers-with-mqtt-broker-bridging.

Django. 2020. Getting started with Django. Série de autoria de Django Project. Retirado a 6 de junho, 2020 de https://www.djangoproject.com/start.

Pycom. n.d. Firmware & API Reference. Série de autoria de Pycom Go Invent. Retirado a 6 de junho, 2020 de https://docs.pycom.io/firmwareapi.

Jay Kapadnis. 2018. REST: Good Practices for API Design. Medium. Retirado a 6 de junho, 2020 de https://medium.com/hashmapinc/rest-good-practices-for-api-design-881439796dc9.

The Flask Project. 2020. Flask: User's Guide. Auditoria de Pallets. Retirado a 6 de junho, 2020 de https://flask.palletsprojects.com/en/1.1.x/#user-s-guide

The Flask Project. 2020. Flask: API Reference. Auditoria de Pallets. Retirado a 6 de junho, 2020 de https://flask.palletsprojects.com/en/1.1.x/#api-reference

Dean Shaff. n.d. Asynchronous vs Synchronous Python Performance Analysis. Stack Abuse. Retirado a 6 de junho, 2020 de https://stackabuse.com/asynchronous-vs-synchronous-python-performance-analysis/

S. Davy, B. Butler, L. Griffin, B. Jennings. 2013. Presentation to OASIS XACML TC concerning JSON-encoded XACML policies. Waterford Institute of Technology. Retirado a 6 de junho, 2020 de https://www.oasis-open.org/committees/download.php/49346/tssgjsonpolicies20130530.pdf

Egor Kolotaev. 2019. Vakt: Attribute-based access control (ABAC) SDK for Python - Documentation. GitHub. Retirado a 6 de junho, 2020 de https://github.com/kolotaev/vakt/blob/master/README.md#documentation

E. Hammer-Lahav. 2010. Request For Comments: 5849 - The OAuth 1.0 Protocol. Internet Engineering Task Force. Retirado a 6 de junho, 2020 de https://tools.ietf.org/html/rfc5849

Universidade de Aveiro. 2012. Protocolo Open Authorization. Universidade de Aveiro. Retirado a 6 de junho, 2020 de https://identity.ua.pt/oauth/

Open Source. 2018. Simple JSON Datasource - a generic backend datasource. GitHub. Retirado a 6 de junho, 2020 de https://github.com/grafana/simple-json-datasource/tree/backend-updated.

Eclipse Foundation. 2019. Documentation :: Eclipse Hono. Retirado a 6 de junho, 2020 de https://www.eclipse.org/hono/docs/

Helder Eijs. nd. Documentation :: AES - PyCryptodome Documentation. Retirado a 6 de junho, 2020 de *https://pycryptodome.readthedocs.io/en/latest/src/cipher/aes.html*

Universidade de Aveiro. nd. Wiki de Componentes - MakerLab. Retirado a 4 de Março, 2020 de https://deti-makerlab.ua.pt/wiki/equipments/

Google. n.d. Flutter - Dart API docs. Flutter SDK. Retirado a 6 de junho, 2020 de https://api.flutter.dev/

8 Leitura adicional recomendada

- **Página de documentação do projeto**
- **Página de Registo Semanal de desenvolvimento do projeto**
- **Página de documentação da API**
- **Página principal do projeto**
- **Página de documentação sobre o efeito da suspensão das atividades letivas devido à COVID-19 no desenvolvimento do projeto**